A FLEXIBLE CONTROL SYSTEM

FOR

FLEXIBLE MANUFACTURING SYSTEMS

A Dissertation

by

WESLEY DANE SCOTT

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2004

Major Subject: Industrial Engineering

A FLEXIBLE CONTROL SYSTEM

FOR

FLEXIBLE MANUFACTURING SYSTEMS

A Dissertation

by

WESLEY DANE SCOTT

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Jeffrey S. Smith (Co-Chair of Committee)	Donald A. Maxwell (Member)
Cesar O. Malave (Co-Chair of Committee)	Sheng-Jen (Tony) Hsieh (Member)
Brett A. Peters (Head of Department)	

May 2004

Major Subject: Industrial Engineering

ABSTRACT

A Flexible Control System for Flexible Manufacturing Systems. (May 2004)

Wesley Dane Scott, B.S., Oregon State University;

M.S.E., Purdue University

Co-Chairs of Advisory Committee: Dr. Jeffrey S. Smith
Dr. Cesar O. Malave

A flexible workcell controller has been developed using a three level control hierarchy (workcell, workstation, equipment). The cell controller is automatically generated from a model input by the user. The model consists of three sets of graphs. One set of graphs describes the process plans of the parts produced by the manufacturing system, one set describes movements into, out of and within workstations, and the third set describes movements of parts/transporters between workstations.

The controller uses an event driven Petri net to maintain state information and to communicate with lower level controllers. The control logic is contained in an artificial neural network. The Petri net state information is used as the input to the neural net and messages that are Petri net events are output from the neural net.

A genetic algorithm was used to search over alternative operation choices to find a “good” solution. The system was fully implemented and several test cases are described.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Page

Page

Page

Page

Page

LIST OF FIGURES

Page

NOMENCLATURE

parts	Items that flow through the factory and will eventually be sold
process plan	An OR graph that describes the production of the part. Processes occur at nodes and have times associated with them. Arcs represent processing constraints
instruction set	Details on how a process should be completed in a format the processor can understand.
material processors (MP)	A piece of equipment that makes changes to the state of a part
material transporters (MT)	Moves parts between physical locations in the factory but can not be used to load a part into a material processor
material handlers (MH)	Can be used to load a part into a material processor
automated storage (AS)	Physical space for long term storage that has a small subset of spaces that are used for interfacing with the rest of the system
buffers (BF)	Physical space for temporary storage
tlocation	A physical space where a transporter can stop
TL	The set of all tlocations
plocation	A physical space where a part can be located
PL	The set of all plocations
transporter	The physical entity on which parts move through the system. Pallets for a conveyor system, AGVs for an AGV system
T	The set of all transporters
part carrier	The physical entity which allows a part to be placed on a transporter
FPL	The set of plocations which do not move. Each element of FPL is associated with either a material processor, buffer or automated storage device
MPL	The set of plocations which move. Elements of MPL are associated with transporters
transportation device	The physical entity that moves transporters
load points (LP)	The set of tlocations where parts are removed from material transporters and placed in workstations. LP is a subset of TL

unload points (UP)	The set of tlocations where parts are placed on transporters and removed from workstations. UP is a subset of TL
transporter movement graph (TMG)	A directed graph that describes the transportation system. Nodes represent physical locations where transporters can stop. Arcs represent possible movements.
transporter movement (TM)	Movement of a transporter from a physical location represented by a node in the TMG to a second physical location represented by a second node. An arc in the TMG has been traversed.
workstation part movement graph	A directed graph that describes how parts enter, leave, and move within a workstation
MPL(LP)	The set of movable plocations currently associated with a given load point. This will change based on the transporter occupying the load point.
MPL(UP)	The set of movable plocations currently associated with a given unload point. This will change based on the transporter occupying the unload point.
incompatible transporter movements	Transporter movements the system can not perform simultaneously.

1 INTRODUCTION

Since the late twentieth century, American manufacturing has been facing two major problems: a shortage of skilled workers in the United States and competition from goods manufactured by workers receiving lower wages in developing nations. A potential solution to these problems is to increase the level of automation in the American factory. Increasing automation allows fewer workers to manufacture more goods and because the worker's salary is spread over a larger number of goods, the labor cost per item is reduced, potentially eliminating the cost advantage of the lower wages in developing nations.

The current efforts in automation are identified as flexible manufacturing systems (FMSs) if they are limited to the shop floor or computer integrated manufacturing (CIM) if they include front office functions including computer aided design (CAD) or computer aided process planning (CAPP). Development of these systems began in the 1970s when automatic material handling systems came into use (Lee, 1994). Flexible manufacturing has been identified as a "national imperative" by Rosenfeld (1992) who believes the average United States (US) manufacturing firm is falling behind its international competitors. Chittipeddi and Wallet (1991) believe that the US trade deficit can not be eliminated without relying on flexible manufacturing.

Flexible manufacturing systems combine the advantages of the traditional flow-line and job shop systems, i.e. they have the efficiency of a flow-line with the flexibility of a job-shop. Products can be manufactured efficiently at low-to-medium varieties and volumes, allowing product mixes and output levels to be changed with minimal losses in productivity (Gupta and Cawthon, 1996, Li and She, 1994, Shinichi and Taketoshi, 1992, Haddock and O'Keefe, 1990, Shukla and Chen, 1996).

This dissertation follows the style and format of *Journal of Intelligent Manufacturing.*

Unfortunately, CIM systems are “virtually out of reach of most of the small companies that could most benefit from CIM,” because no commercial software is available to perform integrated control over the individual shop floor components (Smith and Joshi, 1995). Companies are required to create custom implementations for each manufacturing system requiring experts in manufacturing, manufacturing systems, computer programming and networking. Significant costs and expertise are also required to perform system maintenance or system modifications. This expertise is not readily available in most small companies. Gupta and Cawthon (1996) were told by a product manager at a machine tool company that small companies “haven’t even begun employing NC or CNC. Getting into cells would be too great a technological leap for them.”

In 1987, Naylor and Volz stated software “is the integrated manufacturing problem. The machines, robots, material transports, and so forth exist, but the software needed to tie them together into orchestrated flexible robust systems does not.” Gowan and Mathieu (1996) found the major problems with FMSes were associated with the information flow and control subsystem of the FMS. Liu and Zhang (1998) observe that software to carry out integrated control over individual shopfloor components is not commercially available and “rapid generation of shopfloor control software for integrated control of shopfloors remains a challenge.”

Liu and Zhang (1998) further observe that while various control architectures have been proposed in the literature, with some of them, most notably the NIST control hierarchy (Jones and McLean, 1986) and CIM-OSA, becoming “standard,” none of the architectures are adequate. The architectures “are simply verbose, textual descriptions of the general structure of manufacturing systems. In other words, these qualitative descriptions provide a conceptual view of system decomposition without providing the specific details required to formalize the control software requirements for a control system based on these architectures.”

Simpson *et al.* (1982) believe, “If flexible manufacturing systems are to become widely adopted in the discrete parts industry where 87% of the firms employ less than 50 persons than they are today. It

must be possible for a firm to start with an NC machine, add a robot, add another machine, and so on as capital is accumulated and as the firm's business grows. Systems must also be capable of being tailored to various part mixes without extensive engineering effort." In other words, it must be possible to easily and inexpensively build a control system, and even more importantly, changes to the control system when a new machine or new product is added, must be easy and inexpensive. The ability to add new components is described as "expansion flexibility" by Chryssolouris and Lee (1992). Lawley *et al.* (1997) note that FMS controllers are usually custom developed, highly complex and understood by only a handful of skilled technicians, further "much of the knowledge needed to complete an FMS expansion or modification is not transferred from the vendor to facility personnel or is forgotten by the time an expansion or reconfiguration is required." These issues convert the software controller from the "potentially most capable and flexible system component" into a "major limiting factor in effective FMS deployment."

Senehi *et al.* (1991) have suggested that the goal of research and development in CIM should be to provide the technologies for the creation of automated or semi-automated factories that function efficiently and cost effectively. Given that the machines, robots and material transports have been available for decades and control software is not currently available, research and development efforts need to be directed toward developing good control software construction tools.

The function of a control system can be stated quite simply. The *system state* is mapped onto a set of possible *control actions* to determine the control actions that should be executed. The *system state* is defined by the values of a set of *state variables*. *State variables* describe information about the manufacturing system, such as the number and type of parts in the system and the status of a machine. *Control actions* are actions that can be initiated by the controller and that cause the state of the system to change. Equipment failures cause the system state to change, but are not control actions because they are not initiated by the control system.

The problems start with the implementation of this simple concept. Three things need to be identified: the system state, the possible control actions, and the mapping between the system state and the control actions. Identifying these three items begins with creating a model of the system. One of the difficulties in developing a model is that the choice of modeling technique and the variables used to describe the state of the system are linked. The modeling technique must be chosen so that all of the state information necessary for control is available. Further, the modeling technique must include a method of describing the control actions that can be applied to the system. An adequate system model will allow the control actions and the system state to identified. Unfortunately, the model does not directly provide information about how to map the system state to control actions to achieve a desired result.

The mapping of the system state to control action has typically been created on a human observation and experience basis. The simplest way to record this mapping is to use a *state table*. A *state table* is a complete enumeration of all of the possible states of a system based on the state variables. To record the mapping, the state table is augmented with a set of control actions for each state listed in the table. There may be some states, combinations of state variables, that are impossible to physically achieve. These states may be left out of the state table to reduce the size, since no control actions need to be specified.

Theoretically, any system can be controlled using a state table control system. In practice, the size of the state table becomes prohibitive. The number of possible system states is a function of the state variables that describe the system, $N = \prod_{j=1}^{k} b_j$ where N is the possible number of states, b_j is the number of possible values of state variable j, and k is the number of state variables.

To overcome the state space explosion problem, *rule-based control* can be used. Rules are of the form: IF *a set of conditions* THEN *perform these control actions*. Each rule combines the states that meet the conditions in the IF clause. State table control is rule-based control where each rule applies to

only one possible state of the system. The major problem with this rule-based control is the difficulty in developing good rules. Heuristic scheduling rules were created as an attempt to deal with this problem. Panwalkar and Iskander (1977) presented a summary of 113 dispatching rules. Although dispatching rules can provide optimum schedules for small systems, they are generally inadequate. Drake (1996) reports that the effect of any single dispatching rule varies with system dependent factors and concludes that a generalized solution is not possible. Combining or dynamically changing dispatching rules achieves better performance than using a single rule (Herrman *et al.*, 1995, Storer *et al.*, 1992,1995). A major drawback with dispatching rule research is that it does not deal with the material handling and material transport aspects of a flexible manufacturing system.

Control software can be cast into the dichotomy shown in Figure 1 (Smith, 1992). Generic software is software that can be used for a large class of systems without modification. Implementation specific is split into two categories: automatically generated and hand coded. Automatically generated software is software tailored for each specific application, but that does not require a human programmer to do the coding. The necessary source code is created via a computer program from a description entered by the manufacturing system designer. Hand coded software is software written, debugged, and maintained by a human computer programmer and is the most expensive.

Ideally, a generic control software could be used and no changes would be required to the control software when changes were made to the shopfloor. Drake (1996) observes that a number of researchers have argued that due to the “flexible nature of FMS” and the “inherent differences between systems,” “generic, optimal seeking solutions may be too difficult to resolve in real-time” and alternative analysis mechanisms need to be explored.

If a generic control software can not be achieved then the best remaining option is automatically generated software. When a change is made to the shopfloor (i.e. a new machine or new part type is to be manfactured), the user updates the description of the shop and then a generation program translates the revised description into a new controller. The problem that must be overcome to make this feasible is the creation of an algorithm to generate a good mapping from system state to control actions.

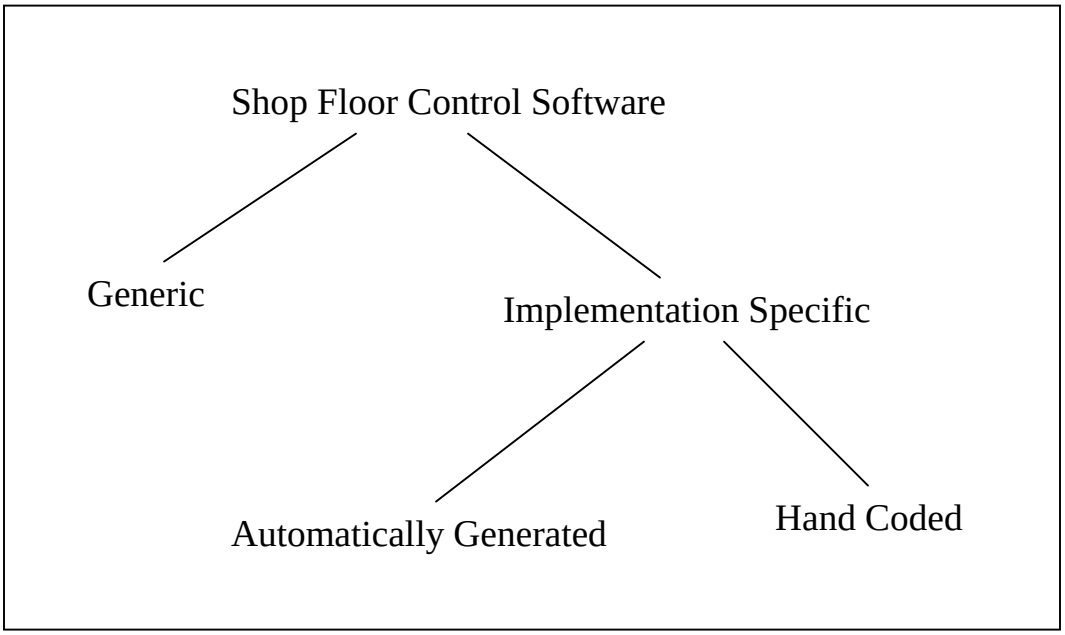

Figure 1 Control Software Dichotomy (from Smith, 1992)

When developing the mapping from system state to control actions, two issues must be considered: safety and performance. Safety consists of three concepts. The first is the elimination of invalid (and potentially dangerous) actions, such as unloading a part when there is no transportation device ready to receive it, causing the part to fall to the floor and be damaged, or trying to load a part onto a machine that is processing another part, causing both parts and the machine to be damaged. The second is to make sure the system is not placed into a state of endless cycling. The third is ensuring that all of the parts to be produced will be produced without placing the manufacturing system into a stalled or

"deadlocked" condition. Lawley *et al.* (1997) claim that "deadlock has emerged as the paramount FMS structural concern."

Cycling occurs when a sequence of control actions is performed and the system returns to a previous state, i.e. no progress is made. Previous work has emphasized producing a completely acyclical system, guaranteeing that any part that enters the system will eventually exit it. This ignores the fact that it may be advantageous for a limited cycle to occur. An example of this case is when a low priority part is moved so a higher priority part can make use of the resources the lower priority part was holding. After the higher priority part has completed processing, the lower priority part would be moved back and reclaim the resources it held before it moved. The low priority part has cycled, but the cell as a whole has not because progress was made by the higher priority part.

Deadlock occurs when system resources are allocated in a manner that will not allow parts to make progress. Coffman *et al.* (1971) identified four conditions that are necessary for deadlock to occur among concurrent processes (each part in a FMS is a process, multiple parts flowing through the system equates to concurrent processes):

1. Mutual exclusion: processes require the exclusive use of a resource
2. Hold while waiting: processes hold onto resources while waiting for additional required resources to become available
3. No preemption: processes holding resources determine when they are released
4. Circular wait: closed chain of processes in which each process is waiting for a resource held by the next process in the chain

Banaszak and Krogh (1990) note that in FMS applications the first three conditions always hold and therefore to avoid deadlocks it is necessary to focus on the fourth condition, a circular wait. They used a simple Petri net model to create a deadlock avoidance algorithm that would guarantee that a circular wait condition would never exist.

The objective of this research has been to demonstrate that flexible manufacturing control systems can be feasibly automatically created. To accomplish this a user friendly manufacturing system model based on three types of graphs was developed. Two of the graph types are used to represent the

workcell in terms of physical locations and the possible movements between physical locations. The third type of graph is used to represent part process plans. These graphs allow the workcell user to define all of the information required to generate the control system eliminating the need for a control engineer to model the system.

The graphs are then algorithmically converted to a particular type of Petri net that is used to interface with other controllers and maintain state information. An artificial neural net is constructed, where the input and output layers are specified by the structure of the Petri net. The hidden layers of the artificial neural net are partially specified by the structure of the Petri net and partially generated as scheduling knowledge is constructed from the process plans and simulation of the workcell performance. The weights of the neural net are constrained so that the structure of the neural net represents logical conditions. Choices among operations are represented by specific weight or node threshold combinations. A genetic algorithm was used to select specific choices. These choices were then implemented by setting the appropriate neural network weight or threshold values.

The dissertation is organized to give the reader a brief review of existing manufacturing system control structures and models and process plan models. Background information on the tools used in this research (Petri nets, artificial neural nets and genetic algorithms) and a discussion of deadlock are then presented in section 2. The specifics of the manufacturing system and process plan models used with a description of the user input requirements are then presented in section 4. The control system and its construction are then described in section 5 followed by the description of a simple system used as a test case in section 6. The results of the work are then presented along with suggestions for future improvements and research possibilities in sections 7 and 8.

2 PREVIOUS RELATED RESEARCH

Related research falls into the following categories: flexible manufacturing control systems, manufacturing system models, process plan models, Petri nets, and artificial neural nets. This research is aimed at developing a manufacturing control system. To develop that control system the manufacturing system and the parts to be manufactured must be modeled. As stated by Adlemo *et al.* (1995), "To be able to control the production efficiently, the controller must have an appropriate model of the manufacturing system, as well as a model for all the products manufactured." To simplify implementation, it appears preferable to use a modeling technique that can be used to model both the manufacturing system and the products produced. Petri nets are such a modeling technique. The literature has examples of Petri nets being used to model manufacturing systems (see Moore and Gupta, 1996, for a review of such models) and process plans. This research uses the Petri net and artificial neural net technologies, applying them in a new manner to the problem of flexible manufacturing system control.

2.1 Flexible Manufacturing Control Systems

Control systems have generally been organized according to one of four models: centralized control, hierarchical control, hybrid control, and heterarchical control. Figure 2 (adapted from Duffie *et al.*, 1988) shows how the control is distributed for the four models. A brief description and an example of each model will be presented.

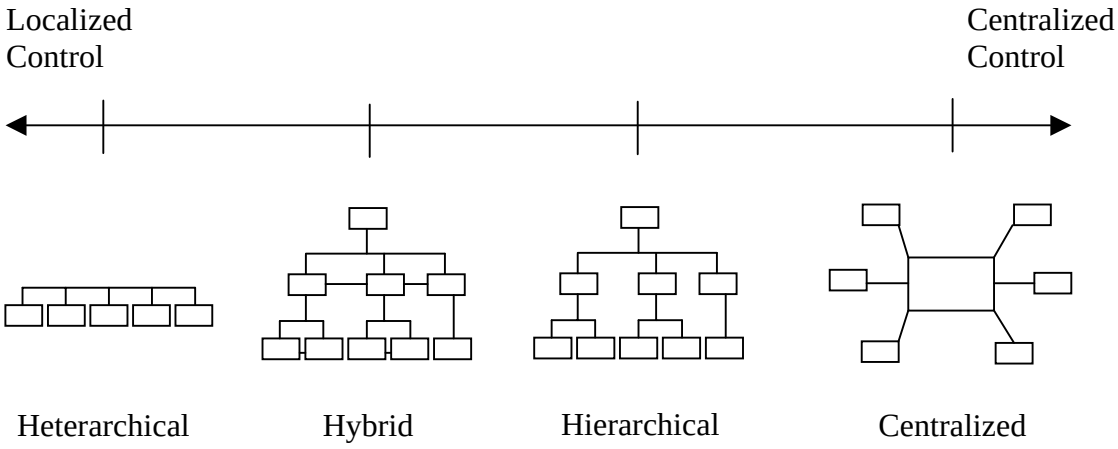

Figure 2 Spectrum of Control Distribution (adapted from Duffie *et al.*, 1988)

2.1.1 Centralized control

Centralized control was implemented in the early period of computer controlled automation. All control decisions were made by one central computer. The major disadvantage of centralized control is the limited size of the manufacturing system that can be controlled. The significant increase in available computing power over the last several decades has reduced the severity of this disadvantage.

2.1.2 Hierarchical control

Hierarchical control was developed to overcome the manufacturing system size limitation of centralized control. In the hierarchical control architecture commands are issued by a central authority figure (computer). These commands are interpreted by the next lower level in the hierarchy where they are either carried out (executed) or detail is added and the commands are passed to the next lower level in the hierarchy, until they reach a level where execution can take place. This architecture is very similar to the standard business organization.

2.1.2.1 National Bureau of Standards (NBS)

One of the first control models was developed at the National Bureau of Standards (NBS) (Jones and McLean, 1986) and applied to the NBS automated manufacturing research facility (AMRF). Based on

an analysis of small batch manufacturing systems they proposed a five level control hierarchy. The levels from the top down were: facility, shop, cell, workstation, and equipment. The facility level deals with “front office” functions and is broken down into three major functional areas: manufacturing engineering, information management, and production management. The shop level has two major components a task manager and a resource manager. The task manager schedules job orders, equipment maintenance, and shop support activities, tracks equipment utilization and schedules preventive maintenance. The resource manager allocates workstations, buffer storage areas, trays, tooling, and materials to cell level control systems for particular production jobs and monitors the levels of raw stock, work in progress, cutting tools, and replacement parts inventories. The cell level is responsible for sequencing batch jobs of similar parts through workstations and supervising the material handling and calibration support services. The workstation level coordinates the activities of small, integrated groupings of physical hardware. The typical workstation in the AMRF consisted of a robot, a machine tool, a material storage buffer, and a control computer. The cell-to-workstation control interface was designed to be independent of the type of workstation. The equipment controllers were “front end” systems tied to a particular piece of equipment. The equipment controller interfaced with the workstation controller and the vendor supplied controller that came with the piece of equipment. The equipment controller translated the workstation commands into a sequence of simple commands the vendor controller can understand. They suggest that it may be possible to partition equipment controllers into two parts: a high level controller that is hardware independent that performs task decomposition, and a low level controller that is hardware dependent that monitors task execution. Controllers were implemented using state tables. They note that a “uniform control architecture” is possible independent of the data required to make a particular part. The process planning system was used to specify not only “all of the machining activities to produce a particular part, but also all robot handling sequences, feasible routings, fixturing, and raw materials.”

2.1.2.2 Wysk and Smith formal functional characterization

Wysk and Smith (1995) describe a shop floor control system (SFCS) with *production requirements* and *resources* as the primary inputs. The output is a set of individual equipment processing instructions that will allow the manufacture and transport of the parts specified in the production requirements. The production requirements consist of administrative and technical requirements. The administrative requirements consist of the number of parts that should be manufactured and are supplied by the shop-wide planning function. The technical requirements include the processing requirements specified by the *process plan* and any special handling or environmental requirements. The process plans are represented as AND/OR digraphs. Resources are non-permanent items such as tooling and fixturing. The process plans for each part that needs to be manufactured are connected via an AND junction to create a composite graph that is used for control. A *task graph* is an AND/OR graph that describes the requirements for individual features of a part. In general, a task graph would correspond to a single node in a process plan.

A *factory model* describes the equipment within the shop and the relationships between the equipment. Their factory model is based on the equipment classification scheme of Smith (1992). A key point is that the factory model is independent of the parts that are produced in the factory.

They suggest that a controller's functionality can be partitioned into *planning*, *scheduling*, and *execution*. Where planning is defined as selecting the tasks the manufacturing system will perform, scheduling is identifying a "good" sequence for performing the tasks based on some performance criteria, and execution is performing the tasks by interfacing with the physical equipment (and possibly other external business systems). They observe that no formal description of the distinction between planning and scheduling has been provided by the research community. Using their formalism of a shop floor control system, planning becomes the "DeOring" of the process plan graph, this represents the selection of a specific set of operations to complete the parts. Scheduling becomes "DeAnding" the process plan graph, this represents selecting the sequence of operations to complete the parts.

2.1.3 Heterarchical control

In the heterarchical control architecture, decisions are distributed. There is no central authority. Each machine determines its next operation based on information available locally. Global optimization can not be performed because no machine knows the complete state of the system. The advantages offered are reduced controller complexity, increased modularity and fault tolerance. These advantages are expected to lead to reduced software development costs and improved maintainability and modifiability.

2.1.3.1 Duffie *et al.* heterarchical control

Duffie *et al*. (1988) demonstrated a system consisting of a machining cell and an assembly cell. The cells consisted of a combination of actual equipment (robots) and simulated equipment (machining stations). They summarize the development process as:

1. Construct initial system using simulated machinery
2. Operate and debug system using simulated machinery
3. Add machine interfaces to an entity
4. Operate system with newly interfaced machine
5. Repeat 3 and 4 until all machines have been interfaced
6. Operate and debug system with actual rather than simulated machinery

They note the ability to mix simulated and actual equipment allows proposed system additions to be studied prior to bringing in the new hardware.

Six design rules were used to “produce a system of cooperating autonomous entities with a high level of intrinisic modifiability and fault tolerance.”

1. Entities should possess the highest achievable level of local autonomy

2. Master/slave relationships should not exist between entities
3. Entities should cooperate with other entities whenever possible
4. Entities should assume that other entities will not cooperate with them
5. Entities should delay establishing relationships for as long as possible
6. Entities should terminate relationships as soon as possible

These principles are based on the principle of minimizing "global information," where global information is defined as any information that is not confined to a single entity. Global information is considered undesirable because "global information and complex relationships between entities makes modification expensive, prone to introduction of logical errors, and often not achievable in the field."

Software for entities in the system was divided into two major components, a controller and a communicator. The controller implemented the control logic and functioned as a state machine synchronized with the hardware associated with the entity. The communicator allowed asynchronous message exchange between entities. The communicator is event driven where events are messages from the network and signals from the entity controller. To achieve fault tolerance, two principles were applied in developing the entities:

1. The entity should not be required to respond to any message it receives; and
2. the entity should assume that transmitted messages will not be responded to by other entities.

The following categories of entities were used: parts, pallets, part processing, material handling robot, and human. Pallets entities contained the part intelligence in the manufacturing system described. Each pallet was responsible for moving through the system according to the plan for manufacturing the parts fixtured to it. Multiple types of pallet entities were required. Part processing entities (machine

tools, assembly robots, inspection stations and input / output stations) are responsible for communicating with other entities, forming relationships with pallets, and translating process requests into detailed sequences of processing control actions for the hardware associated with the entity. Material handling robot entities respond to transportation requests from pallets and recognize the names of stations within their reach. Robot entities are required to coordinate actions if movement is between cells. Human entities were included as advice givers. The human was used to resolve "complex faults," such as, machine failures and "deadlocks caused by 'circular' relationships between part processing and pallet entities." Fault messages generated by other entities in the system were routed to the human entity. After diagnosing the fault, the human would send advice to the entity (e.g. "Continue", "Go to output station") on how to correct the fault.

2.1.4 Hybrid control

Hybrid control is an attempt to obtain the advantages of hierarchical control (potential global optimization) and heterarchical control (redundancy, flexibility) in a single system.

2.1.4.1 Liu and Zhang hybrid control architecture

Liu and Zhang (1998) propose a three level control architecture: shopfloor, agent, and equipment. The equipment level represents a direct mapping of permanent physical equipment, and is the same as that proposed by Smith and Joshi (1995), Jones and McLean (1986), Jones and Saleh (1990), and Cho and Wysk (1995). A formal description of the equipment level is given. "An agent is defined by the aggregated function classes of shopfloor equipment wherever these pieces of equipment are located in the shopfloor." Five types of agents are identified: 1) machining processing (MP), 2) loading/unloading (LU), 3) workpiece-flow (WF), 4) tool-flow (TF), and 5) automated storage (AS). Agents are also categorized into client agents (MP, LU, AS) and server agents (WF, TF). The agent level is defined as

$AL = \{AL_i \mid i = 1,\ldots, n_{al}\}$ where

$AL_i = (AP_i, AU_i, AW_i, AT_i, AA_i)$, a quintuple, known as a multi-agent co-operative cluster.

AP_i, AU_i, AW_i, AT_i, AA_i are couples of the form (AC_i, AG_i) where AC_i is an agent controller and AG_i is an agent (MP, LU, WF, TF, or AS).

A shopfloor SF is defined as SF = {SC, AL} where SC is a shopfloor controller.

Except for the equipment level controllers, the controllers are "independent of the physical structure of the actual shopfloor environment." The shopfloor controller controls the flow of physical material through the shop by assigning tasks to the client agents of a co-operative cluster. The client agents "then request server agents within the same co-operative cluster to provide services and co-operation. Each agent autonomously makes its own decision with its local knowledge base about the shopfloor and controls its relevant equipment." The agent level acts at the same level as the more traditional workstation controller. The differences between agents and workstation controllers are the agent does not control a fixed set of equipment like the workstation controller and agents can communicate with each other, where workstation controllers can only communicate with the controllers above and below them.

2.2 Manufacturing System Models

The Wysk *et al.* (1995) resource model defines resources (R) to consist of equipment (E), tools (T), fixtures (F), transporters (N) and instruction sets (I). The equipment is subdivided into: material processors (MP) which include part transformation equipment and storage, material handlers (MH) which are part transfer devices, and material transporters (MT) devices which "move products from location to another location." Tools are the end-effectors that actually perform a task. Fixtures are devices for "precisely locating and securing a part or set of parts." Transporters are devices for "locating and securing a part or set of parts." Instructions are a "set of commands that instruct a piece of equipment to perform some task." Not considered a resource but defined in the model are ports (P) and locations (L). Ports are subdivided into mports and tports where mports are associated with MP

equipment and tports with MT equipment. Locations are places inside a port where a part can be located. Locations are said to have owners and clients, but a description of how this information is used is not provided. The model includes a graph representation of part movement possibilities (the description of the graph is buried in the definition of the ports). Facilitators are defined as, "A device (from MT or MH) that can move transporters between tports, move parts between tports, or move parts between tports and mports." This definition leaves out the possibility of being able to move parts from machine to machine directly (mport to mport).

Ezpeleta and Colom (1997) partition a FMS into processors and handlers, where processors transform parts and handlers transport parts but do not affect them. Storage systems are considered handlers.

Liu and Zhang (1998) partition equipment (EL) into active (E) and passive (E'). Active equipment requires an equipment controller (EC) and consists of machines with machine controllers (MC). The set EC is partitioned into material processors (EP), material handlers (EH), loading/unloading devices (EM) and automated storage devices (EA). Passive equipment does not require an equipment controller and consists of buffer units that are subdivided into buffers for parts (BP) and buffers for tools (BT). A partial ontology is presented providing a description of the EP and EH classes of equipment. The EP equipment class has two properties, structure (SP) and control (CP). SP has two aspects local part storage capacity (PS) and tool storage (TS). Two types of ports describe the interface to external equipment, part ports (PP) and tool ports (TP). CP describes the exchange mode when interacting with external equipment. Three modes exist: active, the EP equipment controls the exchange; passive, the exchange is controlled by external sources; and interactive, both the EP and the external equipment is involved in the control during an exchange.

For EH equipment the ontology has three properties: the structure property (SH), representing the maximum number of units handled per transaction and the capacity of each unit; the control property, which has the same three modes as the EP class, and the reachability property, the set of locations reachable by the piece of EH equipment.

Activity cycle diagrams (ACD) are constructed to identify controllable activities and interaction processes for equipment controllers. The command sets vary with the structure of the equipment (how the buffers and part ports are arranged).

Adlemo *et al*. (1995) describe a "resource capability model." Resources can be grouped together to create a virtual resource for the next higher level in the hierarchically organized system. Resources are divided into three groups:

1. Producers – these devices make changes to the physical or logical properties of the product, e.g. CNC machines and measurement devices
2. Locations – products are stored, no changes to the products properties are allowed
3. Movers – products are transported between producers and locations e.g. AGVs, robots, conveyors

Comparing the manufacturing system models we find the following commonalities. All models have a category of equipment that produces changes to parts in the system. All models have a category of equipment that moves parts. Wysk *et al*. (1995) subdivide the movement category into material transporters (MT) and material handlers (MH). Liu and Zhang (1998) also subdivide the category using the terms material handlers (EH) and loading/unloading devices (EM). The MT and EH categories and the MH and EM categories appear to be the same.

The point where significant differences occur between the models is the handling of storage equipment and buffers. Wysk *et al.* (1995) include storage systems in the material processor category and do not include buffers (storage without an equipment controller) in the model. Ezpeleta and Colom (1997) also neglect buffers, but place the storage system in the equipment that moves parts (handling) category, not the equipment that changes parts category. Liu and Zhang (1998) include separate categories for automated storage systems (EA) and buffers, where buffers do not require an equipment controller and are subdivided into buffers for parts (BP) and buffers for tools (BT). Adlemo *et al*. (1995) have a location category for part storage.

Additionally, the concept of ports, as places where interaction between categories of equipment occurs, is presented in Wysk *et al.* (1995) and Liu and Zhang (1998).

2.3 Process Plan Models

The function of a process plan is to describe the steps required to transform raw material into a finished product. There is no standardized method of representing a process plan for use with a control system, methods that have been used in the literature include: operations lists, digraphs, AND/OR graphs, Petri nets.

Smith (1992) uses a graph that shows precedent constraints and alternative routings adapted from Metalla (1989). Each node in the graph represents a specific operation or set of operations performed by a machine. Each arc represents movement of the part from the machine represented by the tail of the arc to the machine represented by the head of the arc. Any path through the graph (from start node to finish node) represents a feasible processing route for the part. Hierarchical construction of the graphs showing various levels of detail is proposed with the levels mapping to the hierarchical control structure used. By assigning costs to the nodes and arcs in real-time based on current shop conditions, the shortest path can be used to find an optimum processing route. Smith *et al.* (1992) describes an application of this approach.

Kempenaers *et al.* (1996) discuss the use of non-linear process plans (NLPP) in a collaborative process planning and scheduling system. The system was not intended for use in a fully automated system. The NLPPs provide the scheduler with a set of alternative process plans in an AND/OR graph. An enhanced Petri net model was used to represent the AND/OR graph. Citing others, they report that for constant WIP, productivity can be improved 7.5% and lead-time decreased by 7% by using NLPPs instead of the standard linear process plan. For constant productivity, WIP can be reduced by 25%.

Wysk *et al.* (1995) present a “formal process planning schema” which includes a manufacturing systems resource model. Process plans are represented as AND-OR graphs (a form of NLPP). Nodes

in the graph must be defined in terms of the resource model. "Each node in the graph has an NC file and the associated tooling, fixturing, location, orientation, and processing instructions for creating the feature represented by the node." A description of how this is implemented is not available and the example process plan does not include the information.

Ezpeleta and Colom (1997) model parts with process plans that contain only processors. This contrasts with the working processes of Ezpeleta *et al*. (1995), which describe "the set of possible sequences of operations the system has to perform in order to manufacture a product" and include the material handling operations.

Adlemo *et al*. (1995) describe products by operations lists. The assignment of resources is done by synchronizing a state-machine representing a product operation list with a state-machine representing the resources. The system state is maintained by "a set of concurrently executing state automatons." They state the state information "should be separated from the information that tells the control system what to do when the system has reached a certain state." The "what to do" information is separated into routing and control information. The routing information is created based on the product operation model and a resource capability model. Control information (which is not discussed in the paper) consists of the detailed instructions for the resources, "e.g. which NC programs to run."

2.4 Petri Nets

Petri nets were first described in a Ph.D. dissertation by Carl Petri (1962). The standard references are Peterson (1981), the first book to cover them, and Murata (1989). Many variations have been proposed to the original theory. The most significant of these variations are the addition of deterministic time, Ramchandani (1974), stochastic time, Florin and Natkin (1982), Molloy (1982), color, Jensen and Rozenberg (1991), Jensen (1992), hierarchy and events. Petri nets consist of 4 primitive elements: tokens, places, transitions, and arcs, and the rules that govern their operation.

"A Petri net is a particular kind of directed graph, together with an initial state called the *initial marking*, M_0" Murata (1989). The arcs in the graph have weights associated with them. The weight indicates the number of tokens that must be in the place at the tail of the arc for the transition at the head of the arc to be enabled. An arc with a weight *w* (*w*-weighted) arc is equivalent to a set of *w* parallel arcs with a weight of one. A marking assigns a non-negative integer *k* to each place, where *k* represents the number of tokens contained in the place. A marking is denoted by an *m*-vector, *M*, where *m* is the total number of places in the Petri net. *M(p)*, the *p*th component of *M*, is the number of tokens in place *p* (*M(p)* = *k*). A Petri net is said to be *pure* if it does not contain any self-loops. A self-loop occurs when a place is both an input and an output place for a transition *t*. A Petri net is called *ordinary* if the weights of all of the arcs in the net are equal to one.

Table 1 contains some typical interpretations of transitions and places. In modeling FMSs, input places would represent either preconditions or resources needed. Transitions would represent events or tasks and output places would represent postconditions or resources being released. The interpretations are somewhat interchangeable. Consider a robot that is to load a machine, one can say that for the load operation to take place one needs the resources of a robot, a machine and a part, or one can say that the following conditions must be true, a robot is available, the machine is available and a part is available.

Table 1 Some Typical Interpretations of Transitions and Places (from Murata, 1989)

Input Places	Transition	Output Places
Preconditions	Event	Postconditions
Input data	Computation Step	Output data
Input signals	Signal Processor	Output Signals
Resources needed	Task or Job	Resources released
Conditions	Clause in Logic	Conclusions
Buffers	Processor	Buffers

The dynamic behavior of a system is simulated by changing the marking of the Petri net using the following firing rule assuming each place can hold an infinite number of tokens, i.e., the net is an *infinite capacity net* (Murata, 1989):

1) "A transition *t* is said to be enabled if each input place *p* of *t* is marked with at least *w(p,t)* tokens, where *w(p,t)* is the weight of the arc from *p* to *t*.
2) An enabled transition may or may not fire (depending on whether or not the event actually takes place).
3) A firing of an enabled transition t removes *w(p,t)* tokens from each input place *p* of *t*, and adds *w(t,p)* tokens to each output place *p* of *t*, where *w(t,p)* is the weight of the arc from *t* to *p*."

Nets where the places are limited in capacity are called *finite capacity nets*. Each place has an associated capacity *K(p)*, the maximum number of tokens that *p* can hold at any time. For finite capacity nets an additional condition must hold for the transition to be enabled:

4) The number of tokens in each output place *p* of *t* cannot exceed its capacity *K(p)* after firing *t*.

When all four conditions are included, the firing rule is called the *strict transition rule*. Without constraint 4, the rule is called the *(weak) transition rule*. It is possible to transform a finite capacity net by adding *complementary places* to allow the weak transition rule to be used instead of the strict transition rule.

Petri nets can be used to represent finite-state machines. Petri nets representing finite-state machines are distinguished by the fact that each transition has exactly one incoming arc and exactly one outgoing arc. State machines allow representation of choice (also referred to as conflict or decision), but do not allow the synchronization of activities in parallel. Systems with choice are non-deterministic.

Petri nets that allow representation of *concurrency*, events occurring in parallel, but not choice are called *marked graphs*. Marked graphs are distinguished by the fact that each place has exactly one incoming arc and exactly one outgoing arc. *Confusion* exists when a situation involving both conflict and concurrency occurs.

Moore and Gupta (1996) surveyed the literature to determine what type of automated manufacturing systems had been modeled using Petri nets, what type of Petri nets had been used and what results were available. They found 53 published models, 17 were flexible manufacturing systems. No models incorporated variable process sequencing. Five categories were used for the type of manufacturing system being modeled: Flow shop, automatic transfer line, job shop, flexible manufacturing system, and assembly operations. The five categories were characterized by their scope (the diversity of job types handled) and their scale (total volume of jobs). Six categories of manufacturing of manufacturing elements were identified: workstations (WSs), material handling systems (MHSs), jobs, storage, other resources, and other constraints. Three categories of Petri nets were used: classical Petri nets, timed Petri nets (both deterministic and stochastic nets) and high level or colored Petri nets. Two categories of analysis were identified: qualitative or structural analysis and quantitative analysis. Qualitative analysis deals with the behavioral properties of the untimed Petri net (reachability, boundedness, liveness, reveribility, and coverability), while quantitative analysis deals with performance over time (manufacturing lead time, work-in-progress, machine utilization, MHS vehicle utilization, throughtput and capacity).

The FMS models described appear to have all of the control functions integrated into the Petri nets. No descriptions of the controllers are given. When available performance measures were generated from simulating the Petri net with the exception of Chan and Wang (1993) who use a Markov chain. Chan and Wang were limited to a model of four stations and five parts because of state space explosion.

Moore and Gupta (1996) identify four reasons that Petri nets have not been fully exploited in the domain of flexible and automated manufacturing: 1) using Petri nets to analyze structural properties of the manufacturing system requires use of a class of Petri nets that suffers from state-space explosion, 2) most models represent specific systems, little attention has been given to developing generic models, 3) the theory for composing large models from components has only been developed for

limited classes of Petri nets, 4) classical Petri nets are extremely powerful as a modeling tool, but are difficult to apply to large-scale problems.

Ang and Bundell (1996) used a timed Petri net to control a model FMS consisting of three robots and three pairs of conveyor belts with sensors. Timing information was associated with the Petri net arcs. Transitions were associated with actions and places with events. Each robot and pair of conveyor belts and two sensors were controlled using an AX5216 card inside a 386DX personal computer running Linux. The Petri net controller ran on a SUN Sparc 5 running the Solaris operating system. Communication between the controllers was accomplished via the transmission control protocol (TCP).

To accommodate the potentially very large size of Petri net required to model real world systems in detail the Petri net system used allowed a hierarchical model to be created. Places in the Petri net could be decomposed into child Petri nets. Firing a transition was associated with a sending a command via TCP to a remote controller. Incoming event signals were compared to the set of places that expected token arrivals. Places that expected token arrivals were places that held "virtual tokens." A virtual token resided in a place but was not available to activate the transition following the place until the delay associated with the arc the token had crossed expired.

The Petri net was manually designed so that all known deadlock states were eliminated. They found that their hierarchical system where only places could be decomposed was not flexible enough, because subsystems with multiple inputs and outputs were very common. Their Petri net developer was being redesigned to allow decomposed blocks to begin and end with multiple transitions and their simulator was being extended to allow colored tokens.

2.5 Artificial Neural Nets

The premise for developing artificial neural nets was the observation that humans can do some things that serial digital computers have a difficult time dealing with (e.g. pattern recognition). This led to a

study of the structure of the human brain. "The human brain is made up of a vast network of computing elements, called *neurons*, coupled with sensory receptors (affectors) and effectors" (Bose and Liang, 1996). The brain contains approximately 10 billion neurons and 90 billion cells providing support for the neurons. The neurons interact with each other via synapses with the average neuron receiving signals from thousands of synapses. The neuron cell bodies tend to occur in layers with the outputs of one layer providing inputs to another layer.

The following organizational and computational principles are employed by the brain (Bose and Liang, 1996):

1) Massive parallelism, 2) A high degree of connection complexity, 3) Trainability, 4) Binary states and continuous variables, 5) Numerous types of neurons and signals, 6) Intricate signal interaction, 7) Physical decomposition, 8) Functional decomposition. A large number of simple slow units are used. The units are connected to a large number of other neurons in complex interaction patterns, yielding a huge number of variables. The connection patterns and strengths of the connections are changeable as a result of accumulated experience. The neurons have two states: resting and depolarization (an electrical pulse is traveling the neuron changing the polarization of the neuron). However, the potentials, synaptic areas, ion and chemical density of the brain are continuous and vary continuously in time and space. The brain uses different types of neurons with different signal types. The interaction of impulses at a neuron is non-linear and depends on multiple factors. The brain is organized as a collection of subnetworks. The subnetworks are sets of densely connected neurons. Neurons in the subnetworks are assumed to be only sparsely connected to distant neurons. Specific functions are assigned to specific areas (subnetworks) of the brain.

There are many neuron connection patterns in the human central nervous system. The three major connection patterns are: divergent connections, convergent connections, chains and loops. Divergent connections involve the output of one neuron being transferred to the inputs of many neurons. Convergent connections involve the output of many neurons being connected to the input of a single

common neuron. Chains involve a series of neurons with the output of a given neuron connected to the input of the next neuron in the series. Loops involve a series of neurons arranged as a chain where at some point the output of a neuron is connected to the input of a neuron earlier in the chain.

Modeling the exact performance of the neuron found in the human brain presents a problem that is analytically intractable. To make artificial neural networks practical, simplified models have been used. The first neuron model to obtain wide recognition was that of McCulloch and Pitts (1943). The McCulloch Pitts neuron is a two-state machine. Each neuron (or "cell") has a single output called the *output fiber of the cell*. The output is allowed to branch after the leaving the cell. Each branch must ultimately terminate at the input connection of a cell. The model allows the output of a cell to be directed back as an input to the same cell. Output fibers are not allowed to merge or fuse together. The terminations of the output fibers are one of two types: excitatory and inhibitory.

The cell is a finite state machine and operates in discrete time instants. At each instant, the cell is either *firing* or *quiet*, the two possible states of the cell. Each state has an associated output. The outputs are conveniently labeled *pulse* for the firing state and *no pulse* for the quiet state. Each cell has associated with it a *threshold* that determines the state transition properties of the cell. At time instant *k+1*, the cell will fire if and only if, at time instant *k*, the number of active excitatory inputs equals or exceeds the threshold and no inhibitor input is active. An alternative formulation is to have the cell fire if the difference between the excitation and inhibition exceeds the threshold.

This work was further developed to create threshold logic units (TLUs) with adjustable weights. The TLU has *n* inputs, $x_1, x_2, \ldots, x_n$, and an output y (see Figure 3). There are $n + 1$ parameters, namely the weights ($w_1, w_2, \ldots, w_n$) and a threshold θ. The TLU computes an output value at discrete time instants $k = 1,2,\ldots$, according to Equation 1. The inputs at the current time instant, $x_i(k)$, are used to compute the output value for the next time increment, $y(k+1)$.

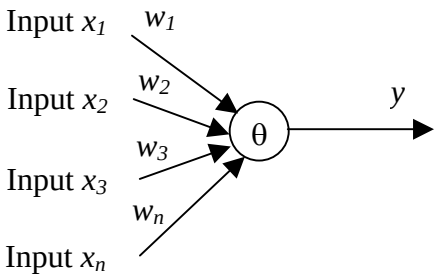

Figure 3 Threshold Logic Unit

$$y(k+1) = \begin{cases} 1 & if \sum_{i=1}^{n} w_i x_i(k) \geq \theta \\ 0 & otherwise \end{cases} \qquad \textbf{Equation 1}$$

where positive weights $w_i > 0$ represent excitatory synapses and negative weights $w_i < 0$ represent inhibitory ones. A bipolar variant of Equation 1 where the zero is replaced by –1 is also commonly used. Another common variation is to use a small positive number (e.g. 0.1) instead of zero as the non-firing output value. This generally speeds up convergence of learning algorithms since it allows the weights connected to the output of the neuron to be updated when the neuron is not firing.

Noting that a real neuron is better described by differential equations than by the discrete time transitions used by TLUs, a neuron model with a continuous transfer function is widely used. This simple model ignores capacitance effects and leakage current in the neuron. The instantaneous input x_i to the *i*th neuron is defined as the mean effects of its excitatory and inhibitory synapses and threshold.

$$x_i = \sum_{j=1}^{n} w_{ij} y_j - \theta_i \,, \qquad \textbf{Equation 2}$$

where w_{ij} are connection weights, y_j is the output of neuron j, θ_i is the threshold of neuron i. The output y_j of a neuron represents the short-term average of the firing rate and is given by:

$y_j = f(\lambda x_j)$, where λ is a positive number.

The transfer function can be defined for the unipolar case or the bipolar case, where

$$y_i = \frac{1}{1+\exp(-\lambda x_i)}$$ is the unipolar, or *logsigmoid* form and **Equation 3**

$$y_i = \frac{2}{1+\exp(-\lambda x_i)} - 1$$ is the bipolar or *tansigmoid* form. **Equation 4**

Both of these transfer functions approach the TLU function as λ approaches ∞.

While the TLU is a very simplified model of a neuron that fails to capture the stochastic spatial and temporal complexities of neuronal information processing, (see McKenna *et al*., 1992, and MacGregor, 1987, for a discussion of other neuron models) it can compute any logical (Boolean) function (Bose and Liang, 1996). FMS control systems are only required to generate a control action when the system state changes. Further, the selection of a control action can be written as a set of logical conditions. The worst-case scenario is one rule for every state the system can occupy, i.e. a state table. Therefore, the TLU neuron model is appopriate for FMS control.

A TLU can be used as either a multi-input "OR," a multi-input "AND" gate or an inverter. These gates are created by adjusting the threshold of the neuron and use input weights of one or negative one. These three types of gates can be combined to represent any Boolean equation. To create an "OR" gate, all input weights are set to one and the threshold is set to 0.5. Figure 4 shows a TLU configured as a two input "OR" gate. If any single input is on (input value equals one), the sum of the inputs will be greater than the threshold and the neuron will produce an output of one. To create an "AND" gate,

all input values are set to one and the threshold is set to *n* minus 0.5 where *n* is the number of inputs to the neuron. The TLU in Figure 4 could be converted to a two input "AND" gate by changing the threshold value to 1.5. If all inputs are on, the sum of the inputs will be greater than the threshold and the neuron will produce an output of one. To create an inverter, a TLU with a single input is used. The weight is set to minus one and the threshold is set to minus 0.5. When the input is zero, the sum of inputs will also be zero and exceed the threshold generating an output of one. If the input is on, the sum of inputs will equal minus one and be below the threshold, so the output will be off.

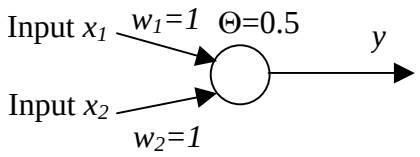

Figure 4 A Two Input OR Gate

Any Boolean equation can be written in the following form: $O = \Sigma (\Pi I_k)$ where I_k can be either an input or the inverse of an input, the Σ represents an "ORing" and the Π represents an "ANDing." Based on this representation a feed forward network with four layers can be used to represent any Boolean equation. The four layers are an input layer, a hidden layer that is used to provide input inverses, a second hidden layer that performs "ANDing," and an output layer that performs "ORing."

Training a neural net consists of systematically selecting a set of weights to achieve the desired outputs for a set of inputs. Training techniques fall into three categories: unsupervised learning, reinforcement learning, and supervised learning. In the unsupervised category, no feedback is given regarding the quality of the output for a given input. In reinforcement learning, general feedback is given about the quality of the solution, e.g. the value of an objective function. In supervised learning, information regarding the error in each output is supplied. Supervised learning requires that a set of known desired outputs be available for each set of input values supplied to the neural network.

Huang and Zhang (1994) examined the use of artificial neural nets in manufacturing. They found neural nets had been applied to the following areas: design, process planning, scheduling, process modeling and control, monitoring and diagnosis, quality assurance, group technology and robotics. Examples include retrieving old product designs that met current requirements, Venugopal and Narendran (1992), Kamarthi *et al.* (1990), Kumara and Ham (1990), Kumara and Kamarthi (1991); predicting the most probable number of forming steps for cold forging Osakada *et al.* (1990), Osakada and Yang (1991a, 1991b); recognizing features Hwang and Henderson (1992); and generating machining operation sequences given a feature Knapp and Wang (1992a, 1992b). An Integral Linear Programming Neural Network (ILPNN) was used to solve job shop scheduling problems formulated as a linear programming problem (Foo and Takefuji, 1988).

Zhang *et al*. (1997) present a method for "automatic induction of parsimonious neural networks." They state, "The search space for neural network induction consists of two levels. One is the space of all possible network architectures (models). The other is the space of all possible weight configurations for a given architecture (parameters)." However, it is not possible to evaluate an architecture without assigning weights and a weight vector cannot be evaluated without knowing the architecture. This makes it necessary to interleave the optimization of the weights and architecture.

They use a tree structure, *NT(d,b)* which denotes the set of all possible trees with maximum depth *d* and maximum branches *b* for each node. The root node is the output unit and the terminal nodes of the tree are the input units. All other nodes are hidden units. The layer of a node is defined as the longest path to a terminal node in its subtree. Two types of neurons were used: sigma units which summed the weighted inputs and pi units multiplied the weighted inputs. Any feedforward network can be represented by using a forest of neural trees. Genetic programming was used to evolve the trees. A crossover operator selected a subtree from Parent B to replace a subtree in Parent A. Local search was used to tune the weights of the network after the structure had been changed. They found that neural trees performed as well as or better than backpropagation networks and required fewer elements.

2.6 Genetic Algorithms

Genetic algorithms were first presented by Holland (1975). Solutions to a problem are represented by strings of alleles (values found at a location) called chromosomes or genomes. A population of solutions is created and each solution evaluated by some fitness measure. Some subset of solutions is then selected to generate a new population of solutions. Solutions are generated using *crossover* where portions of the chromosome from a set of parents is combined to form the a chromosome or *mutation* where a single parent chromosome is randomly changed. The selection of a good crossover operator may mean the difference between a genetic algorithm that works and one that doesn't. A problem with using genetic algorithms for scheduling is ensuring the feasibility of a schedule. If the schedule is used as the chromosome, then a simple crossover operator will not generate a correct schedule. Consider the following four job sequences: A, B, C, D and C, D, A, B. If a simple crossover is performed taking the first half of the first sequence and the second half of the second sequence, the resulting sequence is A, B, A, B. This is clearly not a valid sequence since two of the operations are performed twice and two are not performed at all.

Three common variations of the genetic algorithm are simple (non-overlapping populations), steady-state (overlapping populations) and struggle (overlapping populations) (Wall, 1996). In the simple variation all of the members of the population are replaced with each generation. To prevent the algorithm from forgetting the best solution that has been found, the best individual solution is typically carried forward to the next generation (referred to as elitism). The steady state variation replaces only a portion of the population each generation. The solutions that are replaced are those with the worst fitness factor. This variation converges to a solution faster than the simple solution but is more likely to be trapped in a local minima than the simple variation. The struggle variation is similar to the steady state version but instead of the new solutions replacing the solutions with the worst fitness factor, they replace those with which they have the most similarity. Where a similarity measure (or distance function) represents how different two individuals are either in terms of their chromosome or of their characteristics in the actual solution space.

Storer *et al.* (1992,1995) discuss using meta-heuristics, which include genetic algorithms, based on problem and heuristic spaces. Problem spaces are created by modifying the problem data, e.g. processing times. Heuristic spaces are created by modifying the basic heuristic being used to solve the problem. Two methods of parameterizing the heuristic are presented. The first is based on a weighted combinations of dispatching rules (see Panwalkar and Iskander, 1977) and the second on using one dispatching rule for a fixed number of scheduling decisions and then changing the dispatching rule and using it for the next set of scheduling decisions. They found that using 20 scheduling windows and six heuristics gave good results for problems involving 100 to 500 operations. They used a population size of 50 with 20 percent asexual reproduction (direct transfer of an existing solution) and 80 percent sexual reproduction (crossover from two parents). A mutation probability of 0.15 was used to maintain diversity. For the problem space, the mutation operator added a Uniform(-50,50) deviate to the dummy processing time. For their test problems, they found that a genetic algorithm operating in problem space generated the best solutions.

Hemant Kumar, and Srinivasan (1996) applied a genetic algorithm to solve a static job shop problem using data from a real production shop. The shop had 80 jobs and 59 machines. The number of operations per job varied from 2 to 37. They parameterized the problem using an adaptation of the second method proposed by Storer *et al.* (1992, 1995) as discussed above. Seven dispatching rules were considered. Each rule was used for one scheduling decision and a fixed length string of 10 rules was used. When the number of scheduling decisions was larger than the length of the rule string, the string was restarted. The rule that was used for the scheduling decision was the one occupying position s where s equals (scheduling decision number) modulo n. They reported computer-processing times of less than four seconds for a simple dispatching rule and 998 seconds for their genetic algorithm to schedule a “single batch of 1000 items.” They did not report the type of computer used. After evaluating the fitness function of the initial population of 50 chromosomes, they created a mating pool of 100 chromosomes by randomly selecting from the initial 50 and accepting the chromosome if its acceptance probability (1 – the cumulative distribution of the fitness value) exceeded a randomly

generated number between 0 and 1. The next population was generated using single point crossover (30 percent), two point crossover (40 percent), inversion (28 percent) and mutation (2 percent). The crossover operations are sexual where single point crossover was defined as the exchange of alleles between two chromosomes from a randomly chosen point to the end of the chromosome and two point crossover as the exchange of alleles starting at a randomly chosen point and ending at a randomly chosen point instead of the end of the chromosome. Inversion and mutation are asexual where inversion is the reversal of the order of alleles between two randomly chosen points on a chromosome and mutation as the random interchange of values in two positions. Offspring are created and evaluated for fitness. The offspring are accepted into the next generation if their fitness value is better than the mean of the previous population.

Herrman *et al*. (1995) describe a scheduling system called GAGS (Genetic Algorithm for Global Scheduling). GAGS was applied to an actual semiconductor test facility where schedules are created at the beginning of each 8-hour shift. The semiconductor test facility is a dynamic job shop environment with a rolling horizon that made dividing the decisions into a fixed number of decision windows impractical. Scheduling heuristics were assigned to machines instead of time windows. A *policy* consisted of a combination of heuristics one for each machine. The fitness of the policy was evaluated using a deterministic simulation of the facility. The frequency of the scheduling was limited by the data collection capability of the company's computer integrated manufacturing system, which could only provide work in progress (WIP) extracts once per shift. They found, "the ability to accurately model the test area and automatically compute a shift schedule was just as important to the test area as the ability to find better schedules." Use of GAGS improved on-time delivery from 75-85 % to 90-96% and reduced the time required for creating shift schedules from 120 hours per week to 15 hours per week.

Wall (1996) presents a method of using genetic algorithms for resource constrained scheduling. The genome has two pieces of data at each location: the time to delay after the completion of the last of the

predecessors of the task and the operation mode to use to complete the task. The operation mode represents alternative sets of resources that can be used to complete the task. The method performed best for multi-modal project plans and poorly for job shop problems. The author believed the relative time representation did not work well for the parallel nature of the job shop. The struggle genetic algorithm found better solutions but required more execution time than the steady-state genetic algorithm. The struggle algorithm also always found a feasible solution while the steady-state algorithm did not.

2.7 Deadlock

Wysk *et al.* (1991) argue that deadlock is a significant problem in flexible manufacturing system (FMS) control that "has been ignored by most research in scheduling and control." They note that deadlocks can occur in any "direct-address" FMS, where a "direct-address" FMS employs a "direct-address material handling system such as a robot or a shuttle cart (as opposed to a material-handling system like a recirculating conveyor)." They propose a deadlock detection system based on a graph of "wait relations." They use a string multiplication algorithm to identify circuits in the graph. The algorithm requires that machines be identified by a single character. An M x M (where M is the number of machines) symbol matrix is created and then powers of the matrix are computed to identify circuits in the wait relationships.

Kumaran *et al*. (1994) claim an FMS is a cell level entity in the NIST hierarchical model (Jones and McLean, 1986). However, the model they analyze, four machines, one robot, and a load/unload station is better described as a workstation in the NIST model. They state that "if the number of parts in a system is one less than the number of storage locations, deadlocks can be prevented." They classify deadlock resolution schemes into four categories: (1) conventional, (2) unidirectional batching, (3) deadlock detection and recovery, (4) deadlock avoidance. The conventional scheme uses a large number of storage spaces to prevent deadlock and was not considered because they believed it would increase the work-in-process inventory and transportation costs. The unidirectional batching was not

considered because it would decrease the flexibility of the system. Detection and recovery is a one-step look ahead procedure where the immediate next step is used to determine deadlocks. After a deadlock is identified, one of the parts is moved to a storage location and the remaining parts are moved to their destinations with the part in the storage location then moved to its destination. "A deadlock between any number of machines can be resolved by one buffer." Avoidance is similar to the detection and recovery method, but instead of using only the immediate next step, the entire routings of the parts are considered to avoid impending deadlocks. An impending deadlock is defined as a situation where the immediate transition of parts is possible but the system (or a part of it) will deadlock eventually. They note that while the Wysk *et al.* (1991) procedure works well for detection and recovery it will not avoid all impending system deadlocks because it does not look far enough into the future. They propose an improved version of the procedure in Wysk *et al*. (1991) to be used for deadlock avoidance. They suggest that conservative operation of the FMS may be avoided by allowing deadlock-causing transitions if there is buffer space available to recover from the deadlock.

Leung and Sheen (1993) studied flexible manufacturing cells consisting of "a small number of computer-controlled machines and one or more material handling devices (MHDs)." The cell was assumed to have a central buffer with a capacity of at least two. The central buffer was the only place used for temporary storage of parts. Idle machines did not hold parts unless they were blocked (i.e. the downstream machine for the part was occupied and the central buffer was full). The MHD had a capacity of one. The exit and entry areas were assumed to have infinite capacity and hold parts that are either waiting to enter the cell or have finished processing in the cell. Two deadlock strategies were implemented and compared using simulation. The deadlock avoidance algorithm was said to perform much better than the deadlock detection and recovery algorithm. In the deadlock detection and recovery algorithm one of the buffer spaces in the central buffer was reserved for deadlock recovery. The deadlock detection method was simplistic, requiring all machines in the cell to be blocked simultaneously. The system was then recovered by exchanging the part in the buffer with the part on the machine it was waiting for. The exchanged part was then exchanged with the machine it required

until the entire circular had been resolved. However, if multiple circular waits existed the second or third one would not necessarily be cleared until a part left the system. They note that the buffer space reserved for deadlock resolution will be "fairly underutilized" and that the throughput time decreases as the number of central buffers increases. To improve performance a deadlock avoidance algorithm where the buffer that was reserved for deadlock resolution is allowed to be used if "it is certain that a part from the central buffer (including the reserved space) is going to leave" is proposed. In essence if there is a part waiting in the buffer for the machine that has just finished processing a part then the reserved space can be used for the part on the machine and the part in the buffer moved to the machine to yield a space in the buffer for deadlock resolution. This use of the buffer guarantees that a total system deadlock will not occur. However, it does not prevent "temporary" partial deadlocks. The partial deadlocks will eventually be cleared when a part finally completes processing on one of the machines that is not deadlocked and leaves the cell freeing space in the buffer that can be used for unblocking a machine in the partial deadlock. The policy of not allowing parts to wait idle on machines appears to create deadlocks that could be avoided. If a part completes processing on machine A before parts on machines B and C and no other part currently wants machine A, then placing the part in the buffer creates a partial deadlock if the parts on machines B and C need to exchange places. It also results in unnecessary blocking if the part from machine A needs to go to either machine B or C and the part on that machine needs to go to a machine (other than A) that is occupied. Immediately placing parts in a central buffer when they complete on a machine appears to be a bad policy unless the buffer space is unlimited. If there is infinite space in the central buffer then parts can always leave machines and there will never be a situation where one machine is waiting on another so there will never be a circular wait and the system will never deadlock.

Wysk *et al*. (1994) performed a simulation study comparing two deadlock resolution approaches, avoidance and recovery, to conventional approaches to avoiding deadlock. The study identified the conventional approach as best when the transportation time was low. The authors appear to prefer the avoidance approach because of the potential for "zero in-process inventory and just-in-time

capability," features that are not possessed by the conventional approaches. They note that a system deadlock can be resolved "if there is storage provided to buffer at least one deadlocked part." However, a deadlock situation can be created even with storage if the storage is not properly used.

They categorize approaches to eliminating deadlock into two categories: elimination during system design and elimination through system control. The system design alternatives include unidirectional flow and buffers. Unidirectional flow limits the flexibility of the system but significantly simplifies the control problem. They note that Co and Wysk (1986) have proved that if the number of buffers in the system is one less than the number of parts in the system then deadlocking cannot occur.

System control alternatives include batching and active control, which is partitioned into avoidance and recovery. Batching involves grouping parts and restricting part flow so that flow for each group is unidirectional. In avoidance, the part mix is controlled so that deadlocks are avoided. The procedure of Wysk *et al*. (1991) is used to detect deadlocks. When a new part attempts to enter the system, the routing information of the part and the unprocessed routes of parts in the system are processed. If a potential deadlock is found the part is held at the load station "until it can enter the system without deadlocking all or part of the system."

In recovery, deadlocks are allowed to occur and are resolved by moving parts to buffer spaces reserved for deadlock recovery procedures. The deadlock detection for recovery used only the next immediate destination not the entire routing used in the avoidance algorithm because "routing beyond the immediate destination cannot produce a system deadlock." They randomly chose one of the parts in the identified circular wait to move to a reserved storage and then sequentially moved the other parts based on the part routings.

The conventional approach used for comparison purposes was the use of "large amounts of in-process storage" where the "maximum number of parts allowed in the system is one more than the number of in-process storages." The worst-case situation is then all parts but one are in in-process storage and the

final part can move to any machine it requires. "Deadlock is completely eliminated in this approach," but the authors believe that excessive part transfers and an inefficient manufacturing system will result. In the conventional approach if the next machine required by a part is not available the part is then sent to in-process storage that is always available.

The simulation used included five machines with each part processing on four machines. Statistics collected were makespan, machine utilization and mean flow time. When a machine became available it selected the next part to process in following priority: 1) another machine, 2) in-process storage if it was present, and 3) the input station. In each priority category parts were prioritized by shortest processing time first (SPT). The conventional and recovery methods used the highest priority part. The avoidance approach selected the highest priority part that did not create a deadlock. The avoidance and recovery approaches produced shorter makespans than the conventional approach when the transportation time is greater than twenty percent of the average processing time. They found that the flowtime was always shorter for the avoidance and recovery approaches. This is misleading in that the time spent waiting in in-process storage counts in the conventional approach while time spent in pre-process storage (the load station) does not count in the avoidance approach.

Additional data was collected to study the effect to the number of machines a part was required to visit on the machine utilization. The five-machine system was used with part visiting two, three, four or five machines. They discovered that as the number of machines required to be visited increased, the advantage of the conventional method for low transportation times became larger. When the transportation time became large, the advantage of the avoidance and recovery methods became larger with an increase in the number of machines that had to be visited.

Viswanadham *et al*. (1990) propose a deadlock avoidance procedure using Petri nets. They discuss a procedure for performing deadlock prevention using Petri nets and then determine that it is not feasible for real-world systems. Deadlock prevention is defined as "static resource allocation policies for eliminating deadlock." Deadlock avoidance is defined as "dynamic resource allocation policies."

They state that “deadlock prevention policies that are usually implemented in the design stage lead to inefficient resource utilization. Deadlock avoidance policies that can be enforced during the operation of a system lead to better resource utilization and throughput.” Generalized stochastic Petri nets, a special class of timed Petri nets, are used to model a real-world FMS owned by General Electric and a simple one machine, one automated guided vehicle system. The reachability graph of the simple system is presented. Given the reachability graph a set of resource allocation policies that will prevent deadlock can be determined. The resource allocation policies correspond to selecting the transition that will not lead to deadlock when there is a choice of transitions to fire from a given marking of the Petri net. They note that the reachability analysis can become infeasible if the state space is very large, the situation that prevails in “real-life FMS such as the GE FMS.”

They present an on-line monitoring and control system that will “avoid most of the deadlocks” noting that for deadlocks not predicted “recovery mechanisms have to be used.” They did not actually implement the deadlock avoidance system. They define blocking as a partially enabled transition that has two or more input places. They define a marking as “safe” (noting that safe is inspired by the Operating Systems literature and is not to be confused with “the safeness property” of classical Petri net literature) if it is not blocked or deadlocked. Markings can be “safe,” “blocked,” or “deadlocked.” They define a look-ahead function that identifies the markings that are reachable from the current marking in exactly *i* steps. The controller selects a transition to fire based on the results of the look-ahead function. They made the following observations:

1. “greater look-ahead implies greater probability of avoiding deadlocks. However, there can be systems where only infinite look-ahead will guarantee total deadlock avoidance.” Therefore, deadlock recovery is a necessary supplement to deadlock avoidance,
2. the cost of deadlock recovery decreases with increasing look-ahead,
3. “The PN framework is suitable for implementing deadlock avoidance.”

Banaszak and Krogh (1990) use Petri net models that include “only the aspects of the manufacturing system that are relevant to the deadlock avoidance problem.” They believe their model could be easily extracted from a more comprehensive model of the system noting that other researchers have proposed

Petri net models "for the general specification, simulation, and programming of FMS's." Their FMS consists of a set of "resources" *R* that are modeled with two Petri net places a_r and b_r where tokens in a_r represent available resources of type *r* and tokens in b_r represent busy resources of type *r*. They model parts as a set of operation sequences which are broken up into steps that require only one resource. The sequence of steps is referred to as a *production sequence,* only linear production sequences were allowed. They model the production sequence as a series of places where the first place $p_q(0)$ represents orders waiting to be initiated and the last place $p_q(L_q+1)$ represents completed orders for part *q*. Lq is the length of the production sequence for part *q*. Resource usage is modeled by connecting the transitions in the production sequence to the resource places (a_r, b_r). The combined Petri net is referred to as a production Petri net (PPN). Transitions are *process enabled* if a job is currently in the production step preceding the transition. A transition is *resource enabled* if the place for the resource required for the next step has a token. Deadlock exists when a process enabled transition can never become resource enabled. The precise definition of deadlock used is: "Given sets of resources *R*, products *Q*, and a PPN for the production sequences, a set of transitions $T' \subset T$ is said to be in deadlock for a marking $M \subset R(M_0)$ if 1) all transitions in *T'* are process enabled under marking *M*, and 2) no transition in *T'* is resource enabled for any $M \in R(M)$." They note that transitions in *T'* are not live, in the Petri net sense, but that a transition not being live does not imply that it is involved in a deadlock.

They create a deadlock avoidance algorithm that consists of a restriction policy, where the restriction policy defines a subset of the enabled transitions that allowed to be fired. They note that guaranteeing that there are no transitions that will lead into a deadlock is necessary but not sufficient as the restriction policy ρ could prevent a transition that was both process enabled and resource enabled from firing resulting in ρ*-restricted deadlock*. They leave the selection of the particular firing sequence for the system up to a resource allocation policy that is not covered in the paper. In developing their deadlock avoidance algorithm the note that the production-sequence information for each job in the

system should be used. There is a difference between the way jobs of for the same product and jobs for different products compete for resources. Jobs of the same product with a "straight" pipe where each resource is used only once will not have any conflict for resources. When a resource is used multiple times it is possible to partition the production sequence into subsequences or *zones* which can be treated as individual pipes. Zones are decomposed into subzones using unshared resources and subzones using shared resources. Their restriction policy then consists of two rules: 1) allow a token to enter a new zone only if the capacity of the unshared subzone of the zone exceeds the number of tokens already in the zone, 2) If a shared resource is requested by a job then all of the shared resources in the zone must be available before the job can enter the zone. Three example systems are presented. They suggest possible extensions to liberalize usage of resources, including defining the unshared resources in terms of currently active jobs instead of all possible production routes.

Lawley *et al*. (1997) attempt to "define FMS structural analysis" and provide "guidelines for developing FMS Structural Control Policies, SCP's." An FMS is structurally characterized by its state space (no representation of a state is provided). The state space is represented as a state transition diagram which is a directed graph with states as vertices and state transitions as directed edges. The objective of structural analysis is to "characterize regions of the state space that are structurally sound." Structural control policies (SCP) are then constructed to ensure the FMS operates within a structurally sound region of its state space. State space can not be analyzed enumeratively because it grows exponentially in system size. The deadlock avoidance problem for the resource allocation systems presented (single resource, disjunctive, conjunctive, conjunctive/disjunctive, *k* of *n*) is known to be NP-complete. A structural control policy determines the acceptability of a particular state transition based on the state space structural characteristics. "The SCP should reject any transition leading to a state from which the empty state can not be reached." In general, the obvious solution of applying a search technique to identify a safe sequence (one that will bring the FMS to the empty state) before allowing a transition is not computationally tractable because of the exponential nature of the system state space. Correctly categorizing every state as safe (the empty state can be reached) or unsafe is

generally computationally intractable because of the NP-completeness of the deadlock avoidance problem. An SCP is considered scalable if the computational resource growth is bounded by a polynomial function of system size in terms of the number of jobs and machines. SCPs are required to 1) reject every unsafe state, 2) be scalable, 3) be correct. A *correct* SCP is one the rejects all unsafe states and does not suffer from policy induced deadlock. To eliminate policy induced deadlock, the authors require that for any state accepted by the SCP there must exist a sequence of states acceptable to the SCP that lead to the empty state. An optimal SCP is one that is correct and accepts all safe states. The authors state that optimality is unrealistic and must be "sacrificed for computational tractability." The suggest the ratio of admissible space to safe space as an appropriate measure of the "optimality" of the SCP but state that neither admissible space or safe space is known and must be estimated using simulation.

Generation of a SCP consists of attempting to find a set of necessary but not sufficient conditions "which (1) are present in every deadlock state, (2) present in every unsafe state, and (3) guarantee that for any safe state not exhibiting the condition, there exists a sequence of states not exhibiting the condition which leads to the empty state." The steps for developing an SCP are: 1) Identify a necessary condition for deadlock using some "unique perspective of the FMS," 2) Define the SCP as "An enabled state transition is admissible if and only if the resulting state does not exhibit the necessary condition," 3) Prove scalability, 4) Prove the SCP does not induce deadlock, 5) If a special case FMS was used attempt to extend the SCP to the arbitrary case. A SCP is constructed for "Single Resource Allocation Counter Flow Systems" assuming that machines have a capacity greater than one. The authors state "the most difficult aspect of developing SCP's is identifying the candidate necessary conditions for deadlock states. Unique perspectives such as that of counterflow help provide a basis for deadlock analysis but do not guarantee either unique or suitable necessary conditions. Indeed, it is unclear whether suitable necessary conditions, beyond those already discovered, even exist."

2.8 Summary of Previous Research

The major problem with the previous research can be summed up by applying the concepts in the parable of the Blind Men and the Elephant (see Appendix A). The overall picture is missing, but the individual pieces are relatively well known. The control system models which should give explicit instructions on how to draw an elephant are vague, equivalent to: to draw an elephant, draw a body, add four legs, a tail, and a head with two tusks and two ears. Although, there is debate about how many legs and ears an elephant should have, i.e. should the control system be hierarchical, heterarchical or a mixture of the two.

How should the manufacturing system and the parts that will process through it be modeled? These questions are analogous to asking how an elephant's habitat should be drawn. There are multiple answers to these questions based on the individual researcher's view of what elephants like, but it is important to remember that the researcher has not seen an elephant.

Petri nets are like a paint brush that can be used to draw an elephant (if you have ever seen one) or any of a multitude of creatures and their habitat, but can only be used by a trained artist. Once you have your elephant and its habitat drawn, you can analyze it with a variety of techniques to see what it will be capable of doing in your factory. Unfortunately, if anything is added to the picture drawn with the Petri net brush the analysis techniques become unusable, so Petri net artists generally refuse to add anything to their picture even if it would be a more accurate depiction of an elephant.

Artificial neural nets unlike Petri nets can only be used to describe an elephant (i.e. control system). They cannot be used to describe the habitat (i.e. the manufacturing equipment and parts). The construction of a neural net that describes an elephant is less defined than the construction of a Petri net. It requires an artist of greater skill and perhaps an appreciation of "modern art" on the part of the viewer. Further, what you learn from observing one neural net elephant does not generally teach anything about other neural net elephants.

Genetic algorithms can be difficult to apply to scheduling and control problems because of precedence constraints. The use of heuristic search spaces has made them useful for scheduling when accompanied by a model of the system that is to be scheduled.

The research in deadlock suffers from a problem with the definition of exactly what is an FMS. Is it a cell level construct or is it a workstation? How should storage be handled? The most typical description includes a load/unload station where once a part is placed at the unload station it can never come back. Why? One must seriously ask what is the source of the parts arriving at the load/unload station. Were the parts (raw material) already in the factory or did the raw material arrive with the customers order? If the raw material was in the factory then obviously there was room to store it, so why can't this storage be used later? Remember, if the number of parts is only one greater than the number of in-process storage locations deadlock free operation can be guaranteed (Co and Wysk, 1986) if raw material storage can be used for in-process parts then the number of parts will be less than the available storage and deadlock becomes a non-issue. Further, if a part is allowed to stay at the load/unload station indefinitely before processing has started, why can it not stay there after a subset of the required processes have been completed? This would allow the load/unload station to be used as an in-process buffer, something that is not normally done, but it would allow recovery from deadlocks or at least improve the utilization of the equipment while avoiding deadlocks.

3 PROBLEM STATEMENT

The function of a manufacturing system is to transform raw material into finished products. In the ideal situation, humans are only required to add raw material, to remove finished product and to perform maintenance and repair activities. What separates the flexible manufacturing system from other automated production lines is the ability to rapidly change the product being produced. The unfortunate problem associated with flexible manufacturing systems is that while the product being produced can be changed, it can generally only be changed within a set of parts that were considered when the control system was being developed. A method of including new parts in the set the flexible manufacturing system can produce is needed.

As previously noted, Simpson *et al.* (1982) emphasized the need to be able to build an FMS piece by piece. A likely scenario would be for a firm to purchase a CNC machine and use a human operator. The second step would be to purchase a material handling device (robot) to load the machine from a small local buffer. This would allow the human operator to tend the machine less frequently, giving him more time to perform other tasks. The third step would be to purchase a second material handler to feed parts into the small local buffer from a larger buffer that is beyond the range of the material handler tending the CNC machine. Additional items such as an automated storage and retrieval unit or a material transport device (such as a conveyor system) would then be added as funding became available and experience was developed. Each time a piece of equipment is added the control system becomes outdated and must be replaced or extensively modified.

The method of rapidly generating control software for flexible manufacturing systems described here makes both adding parts to the set the flexible manufacturing system can produce and adding equipment to the FMS practical.

3.1 Verifiable Hypotheses

3.1.1 Factory reference model

A factory reference model can be defined in sufficient detail that its implementation is unambiguous. The existing factory models are limited to conceptual models that do not include enough information to allow implementation in a replicable manner. Two researchers operating from the same factory reference model looking at the same physical system should develop databases that contain the same information. A controller development program built by one researcher should be interoperable with a database developed by a second researcher if both are based on the same factory reference model, assuming a suitable translation program between database programs is available (e.g. from Microsoft Access to MySQL).

3.1.2 Petri net generation

A Petri net model conforming to the formalization in section 5.2 that describes the flexible manufacturing system (FMS) and the parts produced in the FMS can be created from a description of the FMS and parts that conforms to the factory reference model and this Petri net can be used to control the execution of activities in the actual FMS.

3.1.3 Neural net generation and scheduling knowledge creation

A neural net structure that will accept inputs from the Petri net and provide outputs back to the Petri net can be developed. The weights of the neural net can be selected such that appropriate outputs will be generated to cause the Petri net to generate desirable control actions (i.e. control actions that move parts through the system and resolve any stall conditions that occur).

The normal process of selecting the weights of a neural net (supervised learning) requires a set of data that has a set of known outputs paired with a set of known inputs. The net is trained using this known data and then exposed to a set of unknown inputs. In this application (FMS control), the desired outputs for a given set of inputs are unknown. If the mapping of the inputs to the outputs required for

supervised learning was known then there would be no need for this research because a state table controller could be built.

A partial mapping can be determined based on the actions required to move a single part through the system. If a single part is in the system, the state of the part will be represented by a single input neuron that is on (there will be multiple input neurons on if the part is on a part carrier and there are transporters in the system) and a single desired control action can be determined. The input neuron can be connected to the output neuron representing the desired control action through a sequence of links and nodes so that the output neuron will generate a positive output when the input is on.

The neural net is being used to represent logical conditions and their combination using Boolean logic rules. The weights of the neural net will not be changed. Once an element is added to the network its properties are fixed. Any changes required to the logic will be made by adding additional neural net elements (i.e. nodes or links).

3.2 Objectives

To verify the above hypotheses this research has emphasized the following objectives:

1. Propose a standardized interface specification for equipment controllers. Allowing the hand coded portion to be handled by the equipment supplier not the user of the FMS. (Hypothesis 3.1.1)
2. Develop a specification for a user-input description of the manufacturing system and the parts that flow through it and build a database that implements this specification. (Hypothesis 3.1.1)
3. Develop a model for the manufacturing system and the parts that flow through the system that can be used for control and a method of automatically generating these models from the description entered by the user. (Hypothesis 3.1.2)

4. Develop a method for generating control software based on the model of the manufacturing system developed in objective 3. The control software must generate valid solutions, where valid is defined as deadlock and collision free. (Hypothesis 3.1.3)
5. Generating "good" performance without requiring extensive user input. This requires a method of tuning the control software that does not require the FMS user to understand control methodologies. (Hypothesis 3.1.3)

3.3 Test Cases

The test cases selected will be based on the concept of starting small and adding either parts or equipment to the FMS . Test case one consists of a single machine processing workstation and a storage workstation (possibly only a material handler and a set of buffers) with four part types. The main purpose of test case one was to demonstrate the practicality of the Petri net neural net combination. Test case two will add a material transport device to the system where one material transporter will move between two locations. Test case three adds a second machine to the processing workstation and adds additional part routes. Test case four adds a buffer to the processing workstation.

4 MANUFACTURING SYSTEM MODEL

The model used in this research is a derivation of the work done by Smith *et al*. (1996), Smith and Joshi (1995), Smith (1992) and Wysk *et al*. (1995). The major elements in the production system are divided into the following categories: material processors (MP), material transporters (MT), material handlers (MH), automated storage (AS) and buffers (BF). Material processors make some change in either the physical condition (e.g., mills, drills and lathes) of the part or the status of the part (e.g., inspection stations). Material handlers move parts from one position to another in a specified orientation. They are generally used to load and unload material processors. The most common material handlers are various types of robots. Material transporters are used to transport parts to various locations in the factory. Material transporters normally have a larger range of movement than material handlers, but can not be used for loading and unloading material processors. Automated storage has a set of physical spaces for storage. It has an additional set of physical spaces that are used for interfacing with the rest of the production system. Buffers are physical space designated for (usually temporary) storage of parts.

Elements of the manufacturing system are generally combined for purposes of control. This research uses the lower three levels (cell, workstation and equipment) of the set of five control levels defined by Jones and McLean (1986) (facility, shop, cell, workstation, equipment). The functionality of the various levels has been modified. Previous researchers have suggested that each level in a hierarchical system should contain planning, scheduling, and execution functions. This research takes a different view. The workstation and equipment levels are limited to execution functions and all decisions are made at the cell level. This was done for two reasons: in order to achieve a global optimum, global information must be used (particularly where alternative process plan routings involve multiple workstations); and the transportation and storage systems can be used as a large capacity buffer for resolving deadlock.

Global information is particularly important when there are alternative processing routings that involve multiple workstations. The decisions made at one workstation may significantly affect the performance of a second workstation. For example, minimizing flow time in a workstation is achieved by scheduling parts using the shortest processing time for the work that is performed in that workstation. This could easily starve other workstations of work resulting in poor performance or flood a downstream workstation with work it should not have to do. Consider the case where a part has two processing alternatives, it can process on workstation one for 30 minutes or it can process on workstation one for 15 minutes and then be transferred to workstation two to process for 20 minutes. When the part is in workstation one, the decision regarding which process to run should be made at the cell level. If there is a large backlog of parts at workstation one and workstation two is empty it makes sense to transfer the part, if that is not the case then you would want to do all of the processing in workstation one. However, if the workstation was making the decision it would depend on the performance criteria applied to it. For example, if workstation flowtime or makespan were the criteria considered then the workstation controller would select the 15 minute operation to quickly move the part out of the workstation. If the workstation utilization was the criteria then it would select the 30 minute process to maximize the usage of the workstation.

The optimal schedule for minimizing makespan for two machines is achieved by separating the parts into two sets based on the ratio of the processing time on the first required machine to the processing time on the second required machine. The parts in the first set have a ratio less than or equal to one and in the second set have a ratio greater than one. Both sets are scheduled using shortest processing time first and the first set where the ratio is less than or equal to one is processed first. This guarantees that the second machine will experience the minimum amount of schedule-induced delay. The optimum schedule for each machine requires knowledge of the processing requirements on the other machine, i.e. global information.

The following limitations and assumptions have been made for the purposes of this study. Workstations are limited to one material-handling device. The transportation system allows any and all transporters to go to all locations the transportation system serves (the graph of the transportation system is a strongly connected digraph). Material processors have a capacity of only one part. Parts are neither created nor destroyed, i.e. parts must enter and exit process plans only at designated points. Machine setups are either not required or are automated and require zero time. This assumption simplifies the performance evaluation by eliminating sequence dependent setup (or processing) times. The "Prepare to Load" command can be part type dependent. Machine setups would be executed at the equipment controller level in response to the "Prepare to Load" command and are outside the scope of this research.

Workstations are divided into two categories, storage and processing. This research does not use transportation workstations as Smith (1992) did, all transportation control is handled by the cell controller. The control software being developed will handle the movement of the parts inside a processing workstation, but not inside a storage workstation. All of the locations inside a storage workstation will be lumped together and treated as a single location from which parts can be requested. Storage workstations will be treated as if they have a single fixed part location although in reality they will have multiple locations. Storage workstation controllers are outside the scope of this research.

4.1 Parts

A *part* is defined as an individual item that is to be produced by the production system. A part has associated with it a *process plan*. A process plan is an OR graph where each node represents the performance of some operation on the part. This is the same representation used by Smith (1992), Smith and Peters (1998), and Mettala (1989) with the following changes: the current research does not use hierarchical process plans and a node representing raw material is prepended to the graph and a node representing finished product is postpended to the graph. Formally, a process plan $PP_i = <V_i, A_i>$, where V_i is a finite set of nodes representing processing steps for the part and A_i is a finite set of

arcs representing precedence among the processing steps. Figure 5 shows two simple process plans. Plan (a) is for a part with two processing alternatives. Plan (b) is the smallest possible process plan. The minimum size of V_i is 3 (a start node, a finish node, and at least one process node). The minimum size of A_i is 2 (an arc from the start node to the process node and an arc from the process node to the finish node). Associated with each process node in V_i is a *material processor* and an *instruction set*. A material processor is the entity that will perform the operation on the part. The instruction set is the material processor-specific directions on how to perform the operation, typically this will be an NC file.

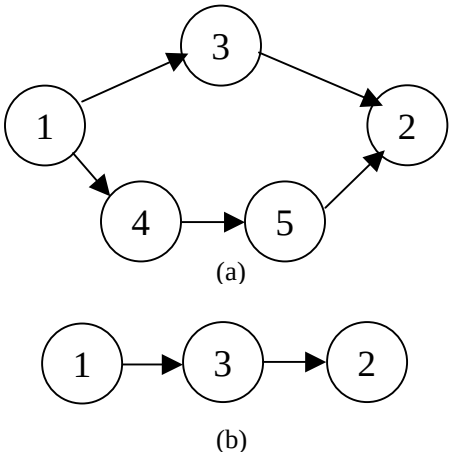

Figure 5 Simple Process Plans

4.2 Manufacturing System

A *transporter* is defined as the physical entity on which parts are moved through the system. Examples of transporters include pallets on a conveyor and automated guided vehicles (AGVs). The set of all transporters will be designated T. A *part carrier* is defined as a physical entity that allows a part to ride on a transporter. The physical characteristics of a part and a transporter will determine whether a part carrier is necessary. A *transportation device* is defined as the physical entity that moves transporters (e.g., the conveyor used to move pallets). A separate transportation device may not be

used by the manufacturing system. Automated guided vehicles serve as both transporter and transportation device, i.e. they move themselves.

A *plocation* is defined as a physical space that can be occupied by a part. The set of all plocations in the factory will be designated **PL**. The set **PL** is partitioned into two disjoint sets: **FPL** and **MPL**. **FPL** is the set of *fixed part locations* and is associated with MP, AS and BF equipment. **MPL** is the set of *mobile part locations*. A mobile part location represents a place on a transporter where a part can be located. Each mobile part location is associated with a specific transporter type.

A *tlocation* is defined as a physical space in the factory where a *transporter* can stop. The set of all tlocations in the factory will be designated **TL**. A *load point* (LP) is defined as a tlocation where parts can be removed from a transporter. An *unload point* (UP) is defined as a tlocation where parts can be placed on a transporter. Load and unload reference the workstation being serviced by the transporter not the transporter, e.g. a part is unloaded from a transporter and loaded into a workstation at a loadpoint. A tlocation can be both a load point and an unload point for the same (e.g. a single tlocation is used for loading and unloading a workstation) or different workstations (an unload point for workstation one is the load point for workstation two).

A *transporter movement graph* describes the possible movements between tlocations and is formally defined as TMG = <**TL**, **A**>, where **TL** is the set of tlocations and **A** is a set of directed arcs describing the possible movements between tlocations. For this study, the TMG was assumed to be either a strongly connected digraph or empty. The TMG can only be empty when all of the tlocations can be accessed by the workstations. Test case one used an empty TMG. All tlocations were occupied by a transporter and all tlocations were both load and unload points; therefore, there was no need for transporters to be moved. A transporter movement (TM) occurs when a transporter traverses an arc and changes tlocations. A transporter movement TM_i is said to be incompatible with a second transporter movement TM_j if the physical transport equipment can not perform the two movements simultaneously. Examples of this would include: a narrow passage where AGVs can not pass each

other going in opposite directions and a conveyor where two movements require the same lift and transfer unit and are transferring in opposite directions.

A *processing workstation* is defined as one or more pieces of MP equipment, one MH device, and zero or more buffers. A *storage workstation* is defined as one or more pieces of AS equipment and one MH device. Associated with a workstation of either type will be a set of load and unload points.

Each workstation will have a *workstation part movement graph*, defined as $WPMG_i$ = <**FPL_i**, **$MPL(LP)_i$**, **$MPL(UP)_i$**, **A_i**), where **FPL_i** is the set of fixed plocations associated with the equipment in the workstation, **$MPL(LP)_i$** is the set of mobile plocations that can occupy the workstation's load points, **$MPL(UP)_i$** is the set of mobile plocations that can occupy the workstation's unload points. A_i is a set of directed arcs that describes the possible movements between the plocations. Each arc has one of three types associated with it (Load, Unload and Transfer) and information regarding whether the movement is limited to a part or whether a part carrier also moves with the part. Data describing the endpoints of the arcs are stored in a from--to format. The meaning of a data item changes depending on the arc type.

Figure 6 shows a workstation movement graph for a simple workstation (it could be a processing workstation or a storage workstation). It is served by two tlocations, one for loading and one for unloading. There is a single fixed part location. The FPL would be a material processor in a processing workstation or the logical fixed part location of an automated storage machine in a storage workstation.

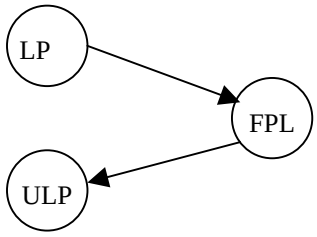

Figure 6 Simple Workstation Movement Graph

4.3 Manufacturing System Activities

An elemental activity is an activity that is performed by a single piece of equipment. The part manufacturing process consists of combining elemental activities to achieve a desired result. Table 2 lists all of the elemental activities that are required assuming that each process command leaves the machine in a state suitable for unloading. If that assumption is not valid, an additional elemental activity, "Prepare to Unload," would have to be added. Table 3 lists the workstation activities. The workstation activities are composed of combinations of the equipment level activities and are described after Table 2 and Table 3.

Table 2 Equipment Level (Elemental) Activities

Task Name	Description
Pick	The material handler will grasp the part and remove it from its current location with no coordination with any other controller.
Place	The material handler will place the part in a location, release the part and move to a clear (safe) location with no coordination with any other controller
Grasp	The material handler will move to a part, grasp it and then wait for further instructions
Take	The material handler will move to a clear location with possession of a part
Put	The material handler will place the part in a specified location without releasing it and wait for further instructions
Clear	The material handler will release any part it is holding and move to a clear location
Prepare to Load	A material processing machine will execute what preparatory action is required to be able to accept a part. The prepatory action may be part type dependent.
Clamp	A material processing machine activates its part holding device
UnClamp	A material processing machine deactivates its part holding device
Process	A material processing machine will load a set of instructions and execute them
Move	A material transport device will move from one tlocation to another

Table 3 Workstation Level Activities

Task Name	Description
Load	A part is loaded from the transportation system into a workstation
Process	Some activity is performed on a part by a material processor
Unload	A part is removed from a workstation and placed into the transportation system
Transfer	A part is moved from one location inside a workstation to another location inside the same workstation

4.3.1 Load

The first step in a load operation is to make sure the destination is ready to receive the part. If the destination is a buffer, no preparation is required. If the destination is a material processor, then a

"Prepare to Load" message must be sent to the material processor. After the destination is ready, a "Pick" command will be sent to the material handler causing the material handler to retrieve the part from the transporter. After the material handler has the part, the next step depends on the destination. If the destination is a buffer, then a "Place" command causes the material handler to place the part in the correct position and release it. If the destination is a material processor, a "Put" command is sent to the material handler causing it to place the part in the correct final position without releasing it. After the part is in place, a "Clamp" command is sent to the material processor. After the part is clamped by the material processor, a "Clear" command is sent to the material handler causing it to release the part and move to a safe position. After the material handler is in a safe position, the load operation is complete.

4.3.2 Process

The process command assumes that a part has already been loaded onto the material processor. The workstation sends a process command to the material processor.

4.3.3 Unload

The unload command assumes that a transporter is at the unload point ready to receive the part. The sequence of commands varies depending on whether the part that is being unloaded is currently located in a buffer or in a material processor. If the part is located in a buffer, a "Pick" command is sent to the material handler, causing it to retrieve the part from the buffer. A "Place" command is then sent to the material handler causing the part to be placed on the transporter.

If the part is located in a material processor, the first step is to send a "Grasp" command to the material handler. The material handler then moves to the material processor and grasps the part, but does not try to remove it. After the material handler has grasped the part, an "Unclamp" command is sent to the to the material processor so it will release the part. A "Take" command is sent to the material handler causing it to remove the part from the material processor. A "Place" command is then sent to the material handler causing the part to be placed on the transporter.

4.3.4 Transfer

The sequence of commands for a transfer command is determined by the type of equipment associated with the plocations involved. The plocation can be associated with either a buffer or a material processor. There are four possible transfers each with its own sequence: buffer to buffer, buffer to MP, MP to buffer, MP to MP. The buffer to buffer sequence is the simplest consisting of a “Pick” command sent to the material handler followed by a “Place” command.

The transfer from a buffer to a material processor is very similar to a load operation. First, a “Prepare to Load” message must be sent to the material processor. After the destination is ready, a “Pick” command will be sent to the material handler causing the material handler to retrieve the part from the buffer. After the material handler has the part, a “Put” command is sent to the material handler causing it to place the part in the correct final position without releasing it. After the part is in place, a “Clamp” command is sent to the material processor. After the part is clamped by the material processor, a “Clear” command is sent to the material handler causing it to release the part and move to a safe position. After the material handler is in a safe position the transfer operation is complete.

The transfer from a material processor to a buffer is very similar to an unload operation. The first step is to send a “Grasp” command to the material handler. The material handler then moves to the material processor and grasps the part but does not try to remove it. After the material handler has grasped the part, an “Unclamp” command is sent to the to the material processor so it will release the part. A “Take” command is sent to the material handler causing it to remove the part from the material processor. A “Place” command is then sent to the material handler causing the part to be placed in the buffer.

The transfer from a material processor to another material processor is the longest of the transfer sequences. The first step is to send a “Prepare to Load” message to the receiving material processor. After the material processor is ready, a “Grasp” command is sent to the material handler. The material handler then moves to the material processor with the part and grasps the part but does not try to

remove it. After the material handler has grasped the part, an “Unclamp” command is sent to the to the material processor so it will release the part. A “Take” command is sent to the material handler causing it to remove the part from the material processor. A “Put” command is sent to the material handler causing it to place the part in the correct final position without releasing it. After the part is in place, a “Clamp” command is sent to the receiving material processor. After the part is clamped by the material processor, a “Clear” command is sent to the material handler causing it to release the part and move to a safe position. After the material handler is in a safe position the transfer operation is complete.

4.4 User Input Requirements

The user must provide the information specific to the facility. Table 4 lists the various data tables that must be completed by the user. The “Parts” table that describes the raw material was implemented as a separate database because of its volatility. Appendix B contains tables that identify the fields contained in each table and the purpose of the field. Tables describing test case one are contained in 6. This research implemented the tables using Microsoft Access.

Because the storage and processing workstation controllers are handled separately, separate but identical tables were used. The tables could have been combined if an extra field had been added to the workstation identification table to indicate the type of the workstation.

Table 4 List of User Input Data Tables

Table Name	Usage
Equipment	Identifies the pieces of equipment in the cell
FixedpLocations	Identifies the fixed places where parts can be located
IncompatibleTransporterMovements	Identifies transporter movements that can not be performed simultaneously
MobilepLocations	Identifies the movable places where parts can be relocated with the transporter type
PartCarrierTypes	Identifies the types of part carriers used in the system and what transporters they can use
PartID	Identifies the various parts the system can manufacture
PPArcs	Process plan arcs representing the manufacturing constraints
PPNodes	Process plan nodes representing the manufacturing steps
ProcessingWorkstations	Identifies the processing workstations in the factory
ProcessingWSEquipAssn	Identifies the equipment contained in each workstation
ProcessingWSLPAssn	Identifies the tlocations where parts can be loaded into the workstation
ProcessingWSMGArcs	Identifies all of the movements that are associated with the workstations
ProcessingWSUPAssn	Identifies the tlocations where parts can be unloaded from the workstation
StorageWorkstations	Identifies the storage workstations in the factory
StorageWSEquipAssn	Identifies the equipment contained in each workstation
StorageWSLPAssn	Identifies the tlocations where parts can be loaded into the workstation
StorageWSMGArcs	Identifies all of the movements that are associated with the workstations
StorageWSUPAssn	Identifies the tlocations where parts can be unloaded from the workstation
TLocations	Identifies the fixed places where transporters can be located
TMGArcs	Identifies possible movements between TLocations
Transporters	Identifies the transporters that are in the system
TransporterTypes	Identifies the types of transporters in the system
Parts	Lists Raw Material in the System

5 CONTROL SYSTEM MODEL

5.1 Organization

As previously stated, this research uses the lower three levels (cell, workstation and equipment) of the set of five control levels defined by Jones and McLean (1986) (facility, shop, cell, workstation, equipment). The levels will now be formally defined starting from the bottom up and brief descriptions of the functions at each level will be provided.

5.1.1 Equipment level

The equipment level in the hierarchy represents a logical view of a machine and an equipment level controller. An equipment level controller and its subordinate machine will be referred to as simply a piece of equipment. Individual pieces of equipment also have machine controllers that provide physical control for the devices. These include CNC controllers, programmable controllers, and other motion controllers and are usually provided by the machine tool vendors. Equipment controllers provide a standard interface (based on the equipment type) to the rest of the control system. This interface hides the implementation-specific code required for machines from different vendors. The equipment controller takes information sent to it from a workstation controller and performs a look-up function to determine what machine specific set of instructions (specific NC file or robot program) must be supplied to the vendor supplied controller. The controller waits for the task to be completed and then sends an appropriate response to the workstation controller.

The equipment controllers are similar to the software/hardware components of Naylor and Volz (1987). They provide a well defined interface (a specific set of formatted messages they will respond to), an internal implementation that is not available to the user, and the controllers are programs that communicate via a TCP/IP connection allowing them to be compiled separately from any workstation level controllers that interact with them. They do require the user to develop machine specific sub-routines that the controllers execute, i.e. NC programs, or robot language movement programs.

Formally, the equipment level is defined as follows (taken from Smith, 1992):

E = $\{e_1, e_2, ..., e_m\}$, is an indexed set of controllable equipment, where:

$e_j \in E$

$e_j = \langle EC_j , D_j \rangle$ where:

EC_j is an equipment controller, and

D_j is a physical device (with device controller).

E is partitioned into {**MP**, **MH**, **MT**, **AS**} where:

MP = $\{e_j \mid D_j$ is a material processor$\}$,

MH = $\{e_j \mid D_j$ is a material handler$\}$,

MT = $\{e_j \mid D_j$ is a material transporter$\}$, and

AS = $\{e_j \mid D_j$ is an automated storage device$\}$.

Unlike Smith (1992) material processing equipment is limited to a single task on a single part. Each piece of equipment in the factory has a capacity, defined to be the number of parts it can hold at one time. The capacity of MP, MH and BF equipment is assumed to be one. The capacity of the MT equipment is equal to the sum of the capacity of the transporters. Smith (1992) also includes a set of equipment known as passive devices that do not use equipment controllers. Passive devices are not considered in this research.

Table 5 describes the formats of the messages the equipment level controllers respond to. Automated storage equipment is only used in storage workstations, which are outside the scope of this research; therefore, a message format for automated storage units was not specified. The messages are parameterized with two types of parameters: run time parameters (rtp) and creation time parameters (ctp). Creation time parameters are known when the control system is being created, eg. transporter locations, part locations. Run time parameters are dependent on the part that is being processed and

are filled in while the controller is running. The parameters column lists the total number of parameters in the message and the number that are filled in at run time. There are two forms of the "Pick" command depending on whether the part is being picked from a transporter in the transportation system or from a buffer in a workstation. There are also two forms of the "Place" command depending on whether the part is being placed on a transporter or in a fixed part location (FPL).

Table 5 Equipment Controller Message Formats

Equipment Type	Message	Parameters	Format String
MT	Move	2 / 0	"MOVE,ctp,ctp"
MP	Prepare to Load	2 / 2	"PREP, TYPE= rtp, NODE= rtp"
	Clamp	0 / 0	"CLAMP"
	Process	3 / 2	"PROCESS, PLOC= ctp, TYPE= rtp, NODE= rtp"
	Unclamp	0 / 0	"UNCLAMP"
MH	Pick from TLocation	4 / 3	"PICK, TLOC= ctp, MPL= rtp, TYPE= rtp, NODE= rtp"
	Pick from FPL	3 / 2	"PICK, PLOC= ctp, TYPE= rtp, NODE= rtp"
	Place in a FPL	3 / 2	"PLACE, PLOC= ctp, TYPE= rtp, NODE= rtp"
	Place on a transporter	4 / 3	"PLACE, TLOC= ctp, MPL= rtp, TYPE= rtp, NODE= rtp"
	Grasp	3 / 2	"GRASP, PLOC= ctp, TYPE= rtp, NODE= rtp"
	Take	0 / 0	"TAKE"
	Put	3 / 2	"PUT, PLOC= ctp, TYPE= rtp, NODE= rtp"
	Clear	0 / 0	"CLEAR"

5.1.2 Workstation level

This research uses the workstation formalism of Smith (1992), but defines only two types of workstations not the three used by Smith. The functions of the transportation workstation defined by Smith have been transferred to the cell level controller. The two types of workstations defined are storage workstations and processing workstations. A storage workstation generally consists of an automated storage (AS) device, a material handler, and an associated set of load and unload points. Storage workstation controllers are outside the scope of this research. The number of locations in a

storage workstation with the corresponding number of movements would cause the size of the controller to increase dramatically for little purpose. Specifying the exact location of a part in the storage workstation will not improve performance significantly and will result in a significant increase in controller size and a corresponding increase in the effort required to train the controller.

An existing storage workstation controller that responds to the interface commands used in this research is assumed. The storage workstation controller responds to requests from the cell controller for parts. It maintains a database of the parts in the storage workstation, selects which part to retrieve if multiple parts of the requested type are available, and sends commands to the material handling equipment and automated storage equipment to retrieve and store parts.

A processing workstation consists of a material handler, a set of load and unload points, and one or more material processing (MP) devices. It may also include one or more buffers (BF). Processing workstation controllers are created as part of the generation of the control logic. The processing workstation controller expands load, unload and transfer commands received from the cell controller and sends the required commands to the material processing and material handling equipment to implement the command. It forwards process initiation commands to the appropriate material processor.

An indexed set of workstations **W** is created, and partitioned into two disjoint sets, a set of storage workstations **WS** = $\{W_1, W_2, ..., W_m\}$and a set of processing workstations **WP** = $\{W_1, W_2, ..., W_r\}$. To accomplish this, the sets **MP**, **MH**, **MT**, **AS**, and **BF**, are each partitioned into subsets indexed by i = 1, 2, ..., m,m+1, m+2, …, m+r, n+1, where *n* = the number of workstations *m+r*, corresponding to the indexing of **W** plus a subset of equipment (n+1) which is not associated with any workstation. For example, **MP** is partitioned into $\{\mathbf{MP_1, MP_2, ... MP_n, MP_{n+1}}\}$. $\mathbf{MP_{n+1}}$, $\mathbf{MH_{n+1}}$, $\mathbf{AS_{n+1}}$, and $\mathbf{BF_{n+1}}$ will be empty sets in this research. $\mathbf{MT_i}$ will be empty unless *i = n+1*, because all transportation equipment is assigned to the cell controller. $\mathbf{MH_{n+1}}$, and $\mathbf{BF_{n+1}}$, may not be empty if the transporter movement graph is not a strongly connected digraph. If the graph is not strongly connected, then some method

must be provided to transship parts between transporters so there will have to be at least one material handler associated with the transportation system and there may also be one or more buffers. A workstation, W_i, is then defined formally as:

$W_i \in \mathbf{W}$

$W_i = \langle WC_i, \mathbf{E_i}, \mathbf{BF_i}, \mathbf{LP_i}, \mathbf{UP_i} \rangle$ where:

WC_i is a workstation controller,

$\mathbf{BF_i}$ is the set of internal workstation storage buffers,

$\mathbf{LP_i}$ is the set of workstation load points,

$\mathbf{UP_i}$ is the set of workstation unload points, and

$\mathbf{E_i} = \{\mathbf{MP_i} \cup \mathbf{MH_i} \cup \mathbf{MT_i} \cup \mathbf{AS_i}\}$.

The workstation controller (WC_i) is created automatically for processing workstations.

Table 6 describes the formats of the messages from the cell controller that the workstation level controllers respond to. Table 7 contains the formats of the messages from equipment controllers that the processing workstation controllers must respond to. The messages are parameterized with two types of parameters: run time parameters (rtp) and creation time parameters (ctp). Creation time parameters are known when the control system is being created, eg. transporter locations, part locations. Run time parameters depend on the part that is being processed and are filled in while the controller is running. The parameters column lists the total number of parameters in the message and the number that are filled in at run time.

Table 6 Workstation Controller Message Formats for Messages from the Cell Controller

WS Type	Message	Parameters	Format String
Processing / Storage	Load	5 / 3	"LOAD,ctp,rtp,ctp, TYPE= rtp, NODE= rtp"
Processing	Unload	3 / 1	"UNLOAD,ctp,ctp,rtp"
Storage	Unload	5 / 3	"UNLOAD,ctp,ctp,rtp, TYPE= rtp, NODE= rtp"
Processing	Process	1 / 0	"PROCESS, PLOC= ctp"
Processing	Transfer	2 / 0	"XFER,ctp,ctp"
Processing	Transform	4 / 0	"TRANSFORM, PLOC= ctp, TYPE= ctp, ONODE= ctp, NNODE= ctp"

Table 7 Workstation Controller Message Formats for Messages from Equipment Controllers

Message	Conversions	Format String
MP prep finished	2 / 2	"PREP, TYPE= rtp, NODE= rtp COMPLETE"
MP clamp finished	0 / 0	"CLAMP COMPLETE"
MP process finished	3 / 2	"PROCESS, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
MP Unclamp finished	0 / 0	"UNCLAMP COMPLETE"
MH Pick from Transporter finished	4 / 3	"PICK, TLOC= ctp, MPL= rtp, TYPE= rtp, NODE= rtp COMPLETE"
MH Pick from FPL	3 / 2	"PICK, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
MH FPL Place finished	3 / 2	"PLACE, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
MH TLocation Place finished	4 / 3	"PLACE, TLOC= ctp, MPL= rtp, TYPE= rtp, NODE= rtp COMPLETE"
MH grasp part in MP finished	3 / 2	"GRASP, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
MH remove part from MP finished	0 / 0	"TAKE COMPLETE"
MH Put finished	3 / 2	"PUT, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
MH clear finished	0 / 0	"CLEAR COMPLETE"

5.1.3 Cell level

The cell level controls the transportation of parts between workstations, the loading and unloading of parts to and from workstations, and the initiation of part processing.

A cell (C) can be formally defined as:

$C = \langle CC, \mathbf{W}, \mathbf{E_{n+1}}, \mathbf{BF_{n+1}} \rangle$ where

CC is a cell controller,

W is the set of workstations previously defined, and

$\mathbf{E_{n+1}}$ is the set of equipment belonging to the cell that is not assigned to a workstation

$\mathbf{E_{n+1}} = \mathbf{MT}$

In this research, the cell controller is formally defined as:

CC = (PN,NN,SM,OV) where

PN is a Petri net as defined in section 5.2

NN is a neural net as defined in section 5.5

SM is a status matrix derived from and updated by the Petri net as defined in section 5.3

OV is an order vector, a user supplied list of parts that should be manufactured as defined in section 5.4.

The outputs of the neural net correspond to the decision events of the Petri net. The input to the neural net is a combination of the status matrix and the order vector. This structure conforms to the belief of Adlemo *et al*. (1995) that state information should be kept separate from "the information that tells the control system what to do when the system has reached a certain state." They also separate the "what to do information" into routing information and control information. Routing information is used to

identify the next resource the part should be sent to. Control information "describes in detail all the actions executed by the resources in each state, e.g. which NC programs to run."

The complete state information is stored in the Petri net. This state information is then translated into a status matrix that is used as an input to the neural net. The neural net stores the information that tells the control system what to do. Routing information is shared between the Petri net, which contains the process plan with its associated precedence constraints, and the neural net, which holds the rules about what to do when a part is in a given state. The "control information" of Adlemo *et al*. (1995) is not included in the cell controller. This information is held at the equipment controller level where a look-up function is performed to determine the set of commands to execute. Minor changes to the product can be made without changing the controller by updating the NC file or replacing it if necessary. Changes to the equipment can also be accommodated, if equivalent programs are available for the new equipment, as long as the basic shape of the process plan does not change and the equipment is located in the same relative location (i.e. is served by a material handler from the same set of load and unload points as the original machine). Referring to Figure 5, if the machine used to process node 4 was replaced (e.g. a three-axis milling machine was upgraded to a four-axis milling machine) no changes would need to be made to the cell controller if an NC program for the new machine existed to process node four and the new equipment had the same equipment controller name as the old equipment. If the shape of the process plan changed, e.g. nodes 4 and 5 are combined into a single node, then the cell controller would have to be changed.

Table 8 contains the message formats that the Petri net portion of the cell controller responds to. The conversions listed are the total conversions and the conversions that are performed at runtime. In general, for each message from the neural net indicating an activity should be started, there is a matching message from a workstation indicating the activity is complete. The exception to this is the "START" command. The "START" command tells the controller that a piece of raw material should be assigned a processing node and specifies the storage workstation where the raw material is located.

Instead of sending a “START” command to the storage workstation, the Petri net sends a “TRANSFORM” command. A “TRANSFORM” complete message is therefore returned from the storage workstation instead of a “START” complete message. The processing workstation unload command does not include information about the type of part to be unloaded. The workstation controller already has information about the part it holds at a plocation; therefore, the neural net does not have to send the information.

Table 8 Cell Controller Message Formats

Message	Source	Conversions	Format String
Load	Neural Net	2 / 0	"LOAD,ctp,ctp"
Load Complete	Workstation Controller	3 / 1	"LOAD,ctp,ctp,rtp COMPLETE"
Unload Processing WS	Neural Net	2 / 0	"UNLOAD,ctp,ctp"
Unload PWS Complete	Workstation Controller	3 / 1	"UNLOAD,ctp,ctp,rtp COMPLETE"
Unload Storage WS	Neural Net	4 / 2	"UNLOAD,ctp,ctp, TYPE= rtp, NODE= rtp"
Unload SWS Complete	Workstation Controller	4 / 2	"UNLOAD,ctp,ctp, TYPE= rtp, NODE= rtp COMPLETE"
Process	Neural Net	1 / 0	"PROCESS, PLOC= ctp"
Process Complete	Workstation Controller	3 / 2	"PROCESS, PLOC= ctp, TYPE= rtp, NODE= rtp COMPLETE"
Transfer	Neural Net	2 / 0	"XFER,ctp,ctp"
Transfer Complete	Workstation Controller	2 / 0	"XFER,ctp,ctp COMPLETE"
Move	Neural Net	2 / 0	"MOVE,ctp,ctp"
Move Complete	MT Equipment Controller	2 / 0	"MOVE,ctp,ctp COMPLETE"
Transform	Neural Net	3 / 0	"TRANSFORM, TYPE= ctp, ONODE= ctp, NNODE= ctp"
Transform Complete	Workstation Controller	4 / 0	"TRANSFORM, PLOC= ctp, TYPE= ctp, ONODE= ctp, NNODE= ctp COMPLETE"
Start	Neural Net	4 / 0	"START, PLOC= ctp, TYPE= ctp, ONODE= ctp, NNODE= ctp"

5.2 Petri Nets

The Petri nets used in this research are defined by a 5-tuple: PN = (P,T,F,M,E), where P is a set of places, T is a set of transitions (some researchers combine P and T into a set of nodes that is later

partitioned), F is a set of directed arcs connecting transitions to places, M is a marking (a description of where tokens are located in the net) and E is a set of events. All arcs have a weight function of one.

The set of places is partitioned into 6 different subsets as described in Table 9. All places except the high capacity place have a capacity limit of one. In the implementation of the Petri net, transitions and places are both derived from class node and are distinguished by their type attribute. The transitions are type one, which is why the partitioning of P begins with type two. All nodes are assigned a time category (see Table 10). Category one is the time between order arrival and starting the part and does not have a corresponding location in the factory. Category zero is used for nodes that are not involved in performance evaluation. All transitions, e-clock places and output places are time category zero. Time information is carried on the tokens (see Table 11, Table 12, and Table 13) and is updated when a token is removed from a place with the exception of time category nine (process plan finished product) places where the time information is updated when the token enters the place. Tokens that enter time category nine places do not exit them. The controller operates in a batch mode so once parts reach the process plan finished product node they remain there until the system is reset for the next batch.

Table 9 Partitioning of P

Symbol	Type	Name	Description
SP	2	Standard Place	General multi-purpose place with capacity one
IP	3	Input Place	Indicates a message has been received from a lower level controller
DI	4	Decision Input Place	Indicates a message has been received from a higher level controller
OP	5	Output Place	Indicates a message should be sent
EC	6	External Clock Place	Indicates the net is waiting for an external event to occur
HC	7	High Capacity Place	A standard place with a capacity greater than one

Table 10 Petri Net Node Time Categories

Time Category	Description
2	Material Transport
3	Material Processing
4	On a Material processor waiting
5	In a Buffer
6	In Storage
7	Material Handling
8	Process plan raw material node
9	Process plan finished product node
0	Node does not apply to a time category

Table 11 Petri Net Token Types

Token Type	Description
0	Information only
1	Transporter
2	Part process plan indicator token
3	A part carrier
4	A part

Table 12 Petri Net Token Data

Token Data Item	Usage
Capacity	The number of other tokens the token can hold
CarrierType	The part carrier type, only valid for token type 3
ListAvailableMPL	The list of available mobile part locations, only valid for token type 1 (tranporters)
MobilePL	The mobile part location occupied by the token
PartType	The type of part if token type is 4, the part type to be selected if the token type is 0
ProcessComplete	The token process status for type 4 tokens or the token process status desired if token type is 0
ProcessNode	The part process plan node if token type is 4, the part process plan node to be selected if token type is 0
PtrListTokensOnBoard	List of pointers to the tokens that are attached to the current token
SMColID	Status matrix column ID
TimeData	See Table 13
TokenType	The token type
TransporterType	The transporter type only valid for token type 2

Table 13 Petri Net Time Data

Petri Net Time Data Item	Usage
AStime	Cumulative time spent in storage after the part has been started and before it is completed
BFtime	Cumulative time spent in buffers after the part has been started and before it is completed
LastEventTime	The time the last event occured
MHtime	Cumulative time spent being moved by a material handler
MPdelaytime	Cumulative time spent on a material processor not processing
MPtime	Cumulative time spent processing
MTtime	Cumulative time spent on a material transporter
OrderArrivaltime	Currently always zero operating the system in batch mode
PartStarttime	The time when the part was started
SimpleFlowEnd	Time the part completes processing on its final machine, used by type 2 tokens
SimpleFlowStart	The time when the part was started, used by type 2 tokens

The arcs contained in F connect transitions and places. An arc originating at a transition must terminate at a place and one originating at a place must terminate at a transition. Each arc has a type associated with it (see Table 14) that determines the type of token (see Table 11) that is allowed to flow along the arc.

Table 14 Petri Net Arc Types

Arc Type	Description
0	Allows all types of tokens to cross it
1	Only allows transporters
3	Only allows part carriers
4	Only allows parts

The preset of a transition (*t) is defined as the set of places where the arcs terminating at the transition originate. The post-set of a transition (t*) is defined as the set of places where arcs originating at the transition terminate. A transition is enabled if all of the places in its preset have the appropriate tokens and the places in its post set have the appropriate space. Determining whether a transition is enabled is complicated by the use of tokens that can contain other tokens or space for other tokens. See the Petri

net marking and token description discussion later in this section. The type of the arcs associated with the transition determines the appropriate tokens and spaces.

A transition will fire as soon as it is enabled if it does not have an event associated with it. This behavior is different from other Petri net research where the firing of a transition may be delayed, the scheduling literature uses the selective firing of transitions to develop schedules (Lee and DiCesare, 1994). If a transition does have an event associated with it, it will be followed by an input or decision input place and will fire when the event occurs, if it is enabled. If the event occurs while the transition is not enabled, the event will be ignored. It is possible for multiple events to be associated with a single message.

The set of events consists of a set of messages that will trigger the firing of an associated transition, assuming all other conditions necessary for it to fire are met. Each event has associated with it, the node number of the transition it fires, the message that will be received, the name of the controller that will be sending the message and the number of data conversions that need to be processed. Events can be divided into two categories based on the relationship of the controller generating the event to the controller receiving the event. Events generated by a controller higher in the hierarchy than the controller receiving the event will be called *decision events*, while events from other controllers will be simply identified as events. The only function of decision events is to place a token in a decision input place. All transitions that react to decision events have an empty preset and a post set that consists of a single decision input place. The preset of non-decision events may or may not be empty. The events associated with process plans will have empty presets while those associated with activities will not.

The Petri net marking describes the location of the tokens in the Petri net. One of the differences between classical Petri nets and this research is that Petri net tokens carry information (see Table 12 and Table 13) and can not be simply destroyed and recreated when a transition fires. Type 1, 2, 3, and 4 tokens represent physical entities in the factory and are conserved across a transition. The tokens are removed from their respective places in the preset and added to the appropriate place in the post set.

Type 0 tokens represent information flow and are created and destroyed as information moves through the workcell. Type 0 tokens are removed from the preset, and any information they carry is extracted and then the token is destroyed. If any type 0 tokens are required in the post set, a new token is created, any necessary information is added and then the token is placed in the appropriate post set place.

While the tokens have types, the nets are not the same as traditional colored Petri nets. Colored Petri nets were developed to reduce the size of traditional Petri nets. By adding color to the tokens and color functions to the arcs that map the token color at the tail of the arc into a token color at the head of the arc, it was possible to combine duplicate node arc structures of the Petri net. The color of tokens in the output places of a transition are determined by the color of the tokens in the input places and the color functions of the arcs to and from the transition. Colored Petri nets can be converted to traditional Petri nets by expanding the network so that each color has its own network structure. In the Petri nets in this research, the token type never changes (equivalent to a constant color function) and it is not possible to eliminate the token types by adding additional network structures.

A significant feature of the Petri nets in this research is that tokens are allowed to contain other tokens. Tokens represent things that travel through the manufacturing system, both physical entities and information. In real world manufacturing systems, some physical entities are associated with other physical entities and travel through the system together. Some method of representing this association must be included in the modeling system (in the simulation domain, the Arena modeling language includes a Group module to combine multiple entities into a single entity, Kelton *et al.*, 1997). The method chosen was to give tokens some characteristics of places and allow them to hold other tokens. This resulted in a more complex transition firing implementation because the firing process must check not only the places in the pre- and post-sets, but also the tokens in the places, for the appropriate type tokens and spaces required to enable the transition. Part carriers (token type 3) can carry parts (token type 4). Transporters (token type 1) can carry part carriers (token type 3) or parts (token type 4). As

previously stated, some parts require part carriers to use transporters, others do not. Information tokens (token type 0 and 2) are not allowed to contain tokens.

Figure 7 shows a common Petri net structure found in this research. It corresponds to an activity. The bars represent transitions. The triangle represents a output place, the square an input place, the large circle a standard place, and the circle with the wedge an external clock place. When conditions are met (the transition's preset is appropriately marked), the initial transition fires placing tokens in the standard place and the output place. The standard place represents the activity in progress and has a row in the status matrix associated with it. The status matrix row will be used for deadlock detection in the cell controller.

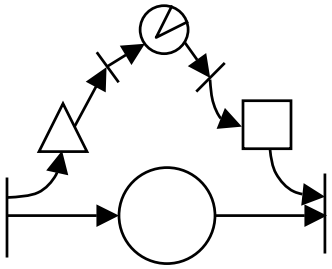

Figure 7 Petri Net Activity Grouping

When the transition following the output place fires, it removes the token from the output message place causing a message to be sent to the destination associated with the output place. It also places a token in the external-clock place indicating that a response is expected from some device. The transition following the external-clock place is an event-triggered transition. (All transitions preceding input places are event-triggered.) When the appropriate message arrives (the message will be from a lower level controller), the transition fires removing the token from the external-clock place and placing a token in the input place. At this point, the final transition may fire depending on the post-set.

At the workstation level, there can be multiple transitions associated with a message from a given controller. For any material processing machine in a workstation, there will be a workstation level

load activity from each load point associated with the workstation and a workstation level transfer activity from the other machines and buffers in the workstation. Figure 8 shows a partial workstation movement graph with two load activities and one transfer activity. Each of these workstation level activities will include an equipment level clamp activity. Each clamp activity will be represented by an activity grouping as shown in Figure 7. The event triggered transition in each activity grouping will be expecting the same message from the same machine, since they all require the same machine to complete the same clamping activity.

When the clamp activity is completed, the machine will send a "Clamp complete" message to the workstation. This message will not indicate the workstation activity that initiated the clamp action. Because event-triggered transitions ignore events that occur when they are not enabled, the workstation controller does not need to specifically track the transition that should be fired when the "Clamp complete" message is received. When the workstation receives the "Clamp complete" message from the machine, it attempts to fire all of the transitions that respond to the "Clamp complete" message. Only the workstation level activity that is being processed will have an enabled transition, i.e. only one of the activities will have a token in the appropriate e-clock place; therefore, the marking of the Petri net can be used to track the activity instead of a separate variable.

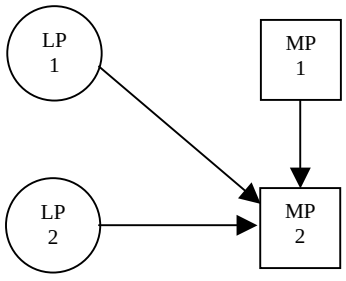

Figure 8 Partial Workstation Graph

5.3 Status Matrix

The status matrix is the method by which the Petri net transfers information to the neural net. The status matrix is also used to determine if the system has developed a deadlock condition. The rows represent locations and activities and the columns represent physical things (transporters, part carriers and parts) that are present at a location or engaged in an activity. There is one row for each fixed part location and each transporter location in the FMS. These rows are not used to detect deadlock. There are two rows for each load or unload activity (these activities involve both a transporter and a part) and one row for each non-load/unload activity controlled by the Petri net. These rows are used to detect deadlock. There is one column for each transporter type, part carrier type and process plan node.

Figure 9 shows a status matrix for a small workcell consisting of three transporter locations, a storage workstation, and a processing workstation with one material processor. The workcell uses one type of transporter, one type of part carrier and makes one type of part, a candlestick with one processing step. The status matrix indicates there is one transporter with a part carrier at transporter location 1 with a finished candlestick, a second transporter is at transporter location 2 with a part carrier and a candlestick that needs to be processed on the material processor. There is a candlestick currently loaded on the material processor ready to begin processing. The storage workstation contains six part carriers, four pieces of raw material and two finished candlesticks.

The rows that represent locations are not used for deadlock detection and are marked false in the deadlock detection column. The rows that represent activities are used for deadlock detection and are marked true in the deadlock detection column. By definition, the workcell cannot be deadlocked if an activity is taking place. However, it is possible for a workstation to be deadlocked while an activity is ongoing outside the workstation. To determine if the workcell is deadlocked, the values of the entries in the rows marked for deadlock detection are summed if the value is greater than zero, then an activity is taking place in the workcell and the workcell is not deadlocked. If no activity is taking place, the number of completed parts is compared to the order vector. If the number of completed parts is less

than the number ordered and no activity is taking place then the system is stalled or deadlocked and corrective action must be taken.

	Transporter	Part carrier	Candle Raw Material Node 1	Candle Node 3	Candle Finished Product Node 2	Deadlock Detection
Tlocation 1	1	1	0	0	1	FALSE
Moving 1-2	0	0	0	0	0	TRUE
Tlocation 2	1	1	0	1	0	FALSE
Moving 2-3	0	0	0	0	0	TRUE
Tlocation 3	0	0	0	0	0	FALSE
Moving 3-1	0	0	0	0	0	TRUE
MP Has Part	0	0	0	1	0	FALSE
Loading MP	0	0	0	0	0	TRUE
UnLoading MP	0	0	0	0	0	TRUE
Load-2 MP	0	0	0	0	0	TRUE
Unload-2 MP	0	0	0	0	0	TRUE
MP Processing	0	0	0	0	0	TRUE
Storage	0	6	4	0	2	FALSE
Loading S	0	0	0	0	0	TRUE
UnLoading S	0	0	0	0	0	TRUE
Load-2 S	0	0	0	0	0	TRUE
Unload-2 S	0	0	0	0	0	TRUE

Figure 9 Sample Status Matrix

The status matrix is updated when a transition fires. The status matrix rows associated with the Petri net places in the transition's pre- and post-sets are updated to reflect the tokens contained in them as part of the transition firing process. Each token stores the index of the status matrix column that represents its identity for use in this update process. Not all transitions will affect the status matrix. The transitions preceding and following e-clock places do not have any Petri net places in either the pre- or post-sets that are associated with status matrix rows.

5.4 Order Vector

The order vector is an organized list of the parts that should be manufactured. It is stored in two data tables. The first table is used to associate the order vector position with a specific part type and is

filled in during the controller creation process. It has one record for each part type the FMS manufactures. The second table is used to store the values representing the number of parts to be created. The table is created during the controller creation process but no records are added to it. The user must manually input the number of parts to be produced.

The order vector is used as an input to the neural net and in the deadlock detection process. When the system determines that no activity is taking place and no new instructions have been issued by the neural net, the number of completed parts in the system is compared to the number that have been ordered (the values stored in the order vector). If the number of completed parts is less than the number that have been ordered then a deadlock or stalled condition is determined to exist.

5.5 Neural Nets

The neural nets used in this research are feed forward neural nets. These nets are also known as back propagation nets because of the way errors are propagated when supervised training is used. This research does not use the typical training methods associated with neural nets. The weights were restricted to the set (-1, 0, 1, 2, 3, …, n), where n is an integer constant defined by the capacity of equipment in the workcell, prohibiting the normal training techniques which assume weights are real valued and continuously variable, normally in the set (-1,1). In developing the deadlock recovery logic, an exception to the weight restriction was made so the network would not need to have an additional hidden layer added.

The net consists of layers of nodes connected via links. Links are directed arcs with associated types and weighting factors. The neural net used in this research uses a single type of node and four types of arcs (see Table 15). Inhibitory arcs have a fixed weight of one. The transfer function used in the nodes is the TLU transfer function (see Equation 1) when the node output is not inhibited. If the node is inhibited, the output is non-pulse (a value of 0.0 is used in this research) for all input values. The network structure is organized to represent a rule based control system. A hidden layer is used to represent logical conditions such as a part of type *t* is in location *l*. Additional layers are then used to

implement a Boolean logic structure, resulting in rules of the form: IF (one or more conditions are true) THEN (activate one or more control actions).

Table 15 Neural Net Arc Types

Arc type	Arc characteristic
0	Excitatory arc with changeable weight
1	Inhibitory arc – inhibit on pulse (high)
2	Inhibitory arc – inhibit on non-pulse (low)
3	Excitatory arc with fixed weight

Rogers (1997) formally describes a neural net as a 3-tuple: NN = (S, P, T), where S is the pattern set, P is the set of network parameters and T is the network topology. The pattern set S = {I,O}, where I is a set of input patterns and O is a set of desired output patterns. The input set I = $\{p_{k,j}\}$, where k is the input pattern number and j is the input pattern component. The output set O = $\{o_{k,j}\}$, where k is the output pattern number and j is the output pattern component.

The parameter set P = $\{p_1,p_2,\ldots,p_n\}$, where p_i is some parameter used in training, testing or operating the neural network. The parameters are generally constants, but can be functions of time or some network characteristic. Common parameters include: learning rate, momentum factor, maximum number of training iterations, and testing tolerance.

The network topology (T) defines the framework (F) and the interconnecting links (L) between the network nodes. T = (F,L). The framework defines the nodes of the network. It is the set of layers (also called clusters) in the network. F = $\{c_1,c_2,\ldots,c_n\}$ A cluster (or layer) is a set of nodes (n) identified by its layer (i) and position (j) within the layer. $c_i = \{n_{i,j}\}$. Nodes are primitive elements of the network. The interconnecting linkage, L = $\{w_{i,j \rightarrow k,l}\}$, defines how the nodes are connected together. A link (w) is identified by the layer (i) and the node position (j) of the starting node and the layer (k) and node position (l) of the terminating node.

The topology of the neural network used in the cell controller is: T = (F,L). F = {c_0, c_1, c_2, c_3, c_4, c_5}, where c_0 represents the input layer, c_1 is a hidden layer, c_2 is a special purpose hidden layer, c_3 a second general purpose hidden layer, c_4 is an initial output layer, and c_5 is an inhibited output layer.

$c_0 = \{n_{0,0}, n_{0,1}, \ldots, n_{0,j}\}$ where j is the number of input nodes, and
j = status matrix rows * status matrix columns + order vector elements.
There is one input node for each cell of the status matrix and one for each element of the order vector (product produced in the cell).

$c_1 = \{n_{1,0}, n_{1,1}, \ldots, n_{1,q}\}$ where *q* is the number of conditions both positive and negative that are used in the Boolean logic.

$c_2 = \{n_{2,0}, n_{2,1}, \ldots, n_{2,r}\}$ where *r* is variable, it includes conditions that can not be represented with a single node in c_1

$c_3 = \{n_{2,0}, n_{2,1}, \ldots, n_{2,s}\}$ where *s* is variable

$c_4 = \{n_{3,0}, n_{3,1}, \ldots, n_{3,k}\}$ where k is the number of output nodes. K is determined by counting the number of decision input nodes in the Petri net portion of the controller.

$c_5 = \{n_{4,0}, n_{4,1}, \ldots, n_{4,k}\}$ where k is the number of output nodes. Layers 4 and 5 use the same number of nodes.

$L = \{w_{0,j\rightarrow 1,l}, w_{0,r\rightarrow 5,s}, w_{1,j\rightarrow 2,l}, w_{1,j\rightarrow 3,l}, w_{2,j\rightarrow 3,l}, w_{3,j\rightarrow 4,l}, w_{4,r\rightarrow 5,r}, w_{4,r\rightarrow 5,s}\}$
The functions of the linkages are:

- ($w_{0,j\rightarrow 1,l}$) generate logical conditions based on the input values
- ($w_{0,r\rightarrow 5,s}$) inhibit incompatible transporter movements if one is already moving
- ($w_{1,j\rightarrow 2,l}$) generate complex logical conditions
- ($w_{1,j\rightarrow 3,l}$) generate “ANDed” logical conditions
- ($w_{2,j\rightarrow 3,l}$) generate “ANDed” logical conditions or implement inhibit choice points
- ($w_{3,j\rightarrow 4,l}$) transfer “ANDed” results to the output layer
- ($w_{4,r\rightarrow 5,r}$) drive the final output layer
- ($w_{4,r\rightarrow 5,s}$) prevent incompatible transporter movement from starting

Not all nodes will be connected. There is an input (layer 0) node for every cell in the status matrix. There are a number of cells that will always be zero and therefore provide no information. The input nodes representing these cells may not be linked to any other nodes. Example cells include all those cells in a row representing a transformation activity in progress that do not represent the part being transformed, i.e. the row represents Part type 2, node 3 being transformed to node 2, all columns that are not Part type 2, node 3 will always be zero.

5.5.1 Choice points

The structure shown in Figure 10 is known as a choice point. It describes the situation where a single condition being true would allow multiple conflicting outputs to be true. When the layer 1 node is true, all of the outputs represented by the layer 4 nodes are valid. However, choosing one of the outputs makes the others invalid. Two examples of this are: a workstation with parts ready to unload on 3 machines all served by the same robot, unloading any of the three machines is possible, but it is impossible to simultaneously unload more than one machine and a part with multiple processing alternatives, choosing one processing path means the other can not be chosen. To ensure that only one of the choices is selected the thresholds of the nodes inside the box are adjusted so that only one of the nodes will be active.

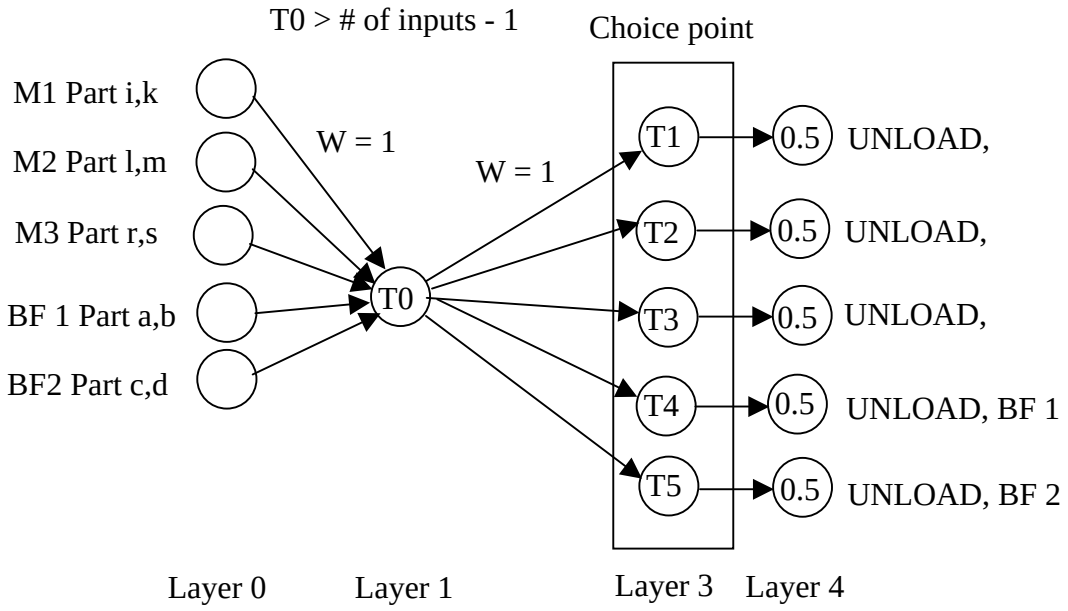

Figure 10 Neural Net Choice Point

5.5.2 Inhibit choice points

The structure shown in Figure 11 is known as an inhibit choice point. The inhibit choice point is used where multiple conditions generate outputs that are incompatible with each other. The layer 3 nodes that are driving the incompatible outputs (layer 4 nodes) are determined. Although it is not shown in Figure 11, it is possible to have multiple layer 3 nodes connected to a layer 4 node. When multiple layer 3 nodes are connected to the layer 4 node only those layer 3 nodes that are on are included. The layer 1 conditions associated with all of the layer 3 nodes that are on are then determined and combined into a new layer 2 node. The output of this new node is used to inhibit all but one of the layer 3 nodes that were on by connecting the layer 2 node to the layer 3 nodes with type 1 (inhibit high) arcs. The arc to the uninhibited node is assigned a weight of zero.

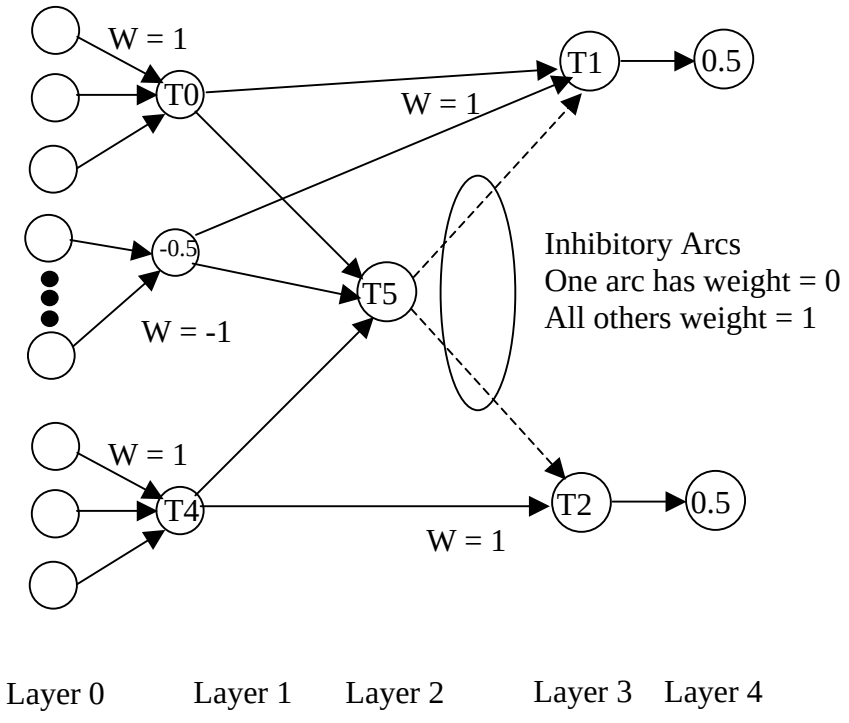

Figure 11 Neural Net Inhibit Choice Point

5.6 Control System Construction

The control system consists of three levels: equipment, workstation, and cell. The equipment level controller construction is out of the scope of this research. Each equipment level controller must be developed individually to fit the piece of hardware. The equipment controller must respond to the messages listed in Table 5.

5.6.1 Construction of the processing workstation controllers

The Petri nets for the processing workstation controllers are constructed using the processing workstation movement graphs. The workstation controller deals with the following activities: loading a part into the workstation, unloading a part from the workstation, moving a part within the workstation, and performing a processing step on a part. The workstation controllers do not attempt to verify that a part is available to load or a transporter is available when a part is to be unloaded. The cell controller is responsible for ensuring that the part or transporter is in the appropriate place before sending a load or unload command to the workstation. See section 5.6.2.3 for a discussion of how the cell controller logic to accomplish this is developed. Figure 12 shows the process of creating the workstation controllers.

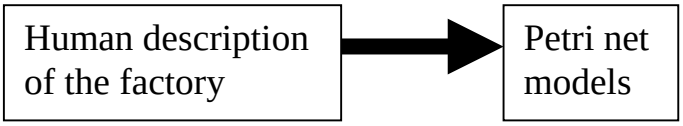

Figure 12 Workstation Controller Construction Process

For each processing workstation, the following seven steps are performed to create the Petri net: (1) an empty Access database with all of the appropriate tables is created, (2) the controller name information is added to the database, (3) a node is added to indicate the availability of the material handling device, (4) for each fixed part location, a node to indicate the FPL is available and a node to indicate a part is

occupying it is added, (5) a processing activity is added for each MP device in the workstation, (6) for each arc contained in the workstation movement graph the appropriate series of activities is added, (7) tokens are added to the places that indicate the material handler and material processors are available. Appendix E illustrates the growth of the Petri net for the simplest processing workstation that can be created (the load point and unload point can be the same physical transporter location).

5.6.2 Construction of the cell controller

Figure 13 shows the process of building the cell controller. The process consists of creating a Petri net to represent the workcell, and the part process plans, extracting information generated during Petri net creation to form the status matrix and the order vector, creating an initial neural net, creating example data for the neural net, using the example data to modify the neural net. The dashed lines represent the flow of information when adjusting for deadlock. If the system does not deadlock (e.g. test case one) the processes represented by the dashed lines would not be used.

5.6.2.1 Petri net

Conversion of the human description into a Petri net is the first step in the cell controller creation process. The Petri net used in the cell controller is different from the Petri net used in the workstation controller. The activities in the cell controller are simplified compared to the activities in the workstation controller and the cell Petri net includes process plan information that is not included in the workstation controller. The Petri net creation process consists of the following thirteen steps:

1. an empty database with all of the appropriate tables is created
2. the controller name information is added to the database
3. add two nodes for each transporter location to represent the location is available and the location is occupied by a transporter
4. a node is added for each material handling device to indicate availability
5. for each fixed part location, add a node to indicate the FPL is available and a node to indicate a part is occupying it, if the equipment associated with the FPL is an automated storage system high capacity nodes are used instead of the standard nodes that are used for the other FPLs
6. add a processing activity for each MP device in the cell
7. add an activity for each transportation movement graph arc
8. add an activity for each processing workstation movement graph arc, this is a single activity not the expanded list of activities that was added in the workstation controller

9. add an activity for each storage workstation movement graph arc
10. tokens are added to the places that indicate the material handler and material processors are available
11. for each transporter, add an entry to the appropriate tlocation has a transporter node, add a token to the tlocation available node for all tlocations that do not have a transporter
12. for each process plan node, add a high capacity node to the Petri net
13. for each process plan arc, add an activity, for each arc leaving a raw material node add a decision input place, for each non-raw material node with multiple arcs leaving it add a high capacity place, for each arc leaving a non-raw material node with multiple arcs, add a decision input place.

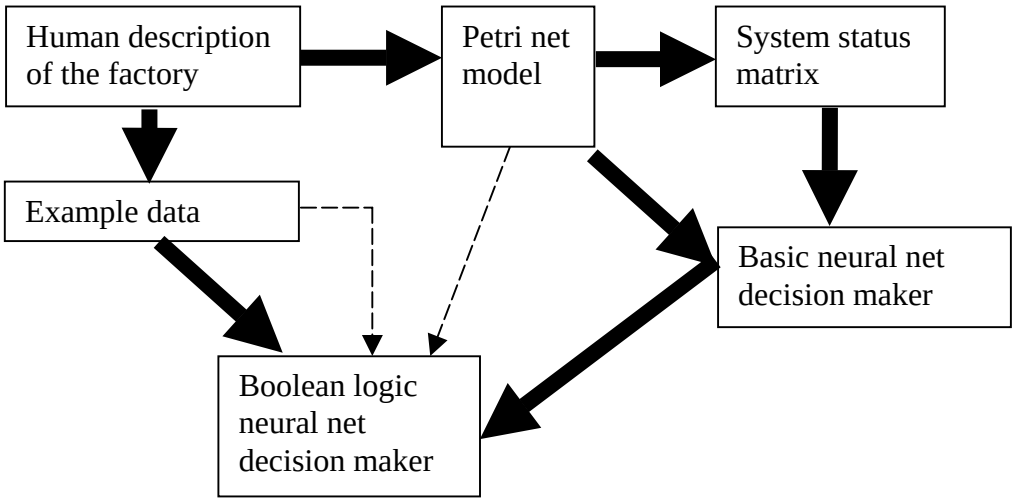

Figure 13 Cell Controller Construction Process

5.6.2.2 Status matrix and order vector extraction

After the Petri net is complete, the status matrix can be created. The information for the status matrix is stored as a table describing the rows and a table describing the columns. During the Petri net creation process, the nodes are marked if they belong to a status matrix row or a status matrix column. The algorithm is presented in Appendix I.

The status matrix column information is created in three steps. The first step is to add an entry for each transporter type that is listed in the user model. The second step is to add an entry for each part carrier type that is listed in the user model. The third step is to add an entry for each of the Petri net

nodes in the PNNode table that are marked as being status matrix columns. These nodes represent the parts in their various stages of manufacture. Information for these entries is filled in using the PartIndex table.

The status matrix row information is extracted from the PNNode table. An entry is added for each node that is marked as being a status matrix row.

The order vector is created by adding an entry for each part type that is listed in the user model. This entry consists of the order vector position and the part identification number, it does not include the number of parts to be produced. For further discussion of the order vector refer to section 5.4.

5.6.2.3 Neural net creation

The initial neural net is created in three steps. The first step is to add a node to the input layer for each entry in the order vector and each entry in the status matrix. The second step is to add nodes to the preliminary and final output layers for each of the various types of commands. The output layers have a node for each valid message the neural net must send to the Petri net portion of the controller. The nodes in the preliminary output layer are linked to corresponding nodes in the final output layer with a fixed weight excitatory link. The third step is to add inhibitor links to prevent incompatible transporter movements from occurring at the same time. Links are added from the input layer to the final output layer to prevent an activity from being initiated if an incompatible activity is already occurring and from the preliminary output layer to the final output layer to prevent two incompatible activities from being started.

Neural net elements are then added to implement a set of rules based on the Petri net decision places. Each decision place in the Petri net has an associated transition that should fire when the decision is issued. Conversely, if the transition cannot fire, (i.e. either the pre- or post- conditions are not met) the message that constitutes the decision should not be sent. To prevent the message from being sent, one or more hidden layer nodes are created with a set of fixed weight excitatory arcs that correspond to the

logic necessary to determine if the transition can fire. The appropriate hidden layer node is then connected to the output layer node with an inhibit on non-pulse arc. The rules used are weak because only one node is used for each decision input place instead of one for each neural net output. Figure 14 shows a portion of the logic for a load decision input place (the robot and destination availability logic is not shown). All of the messages on the right side of the figure will cause a token to be placed in the decision input place associated with the rule. If any of the conditions on the left are true, then the layer one node will generate a positive output. This means that a part of type one node six at Tlocation 1 will eliminate the inhibition on all of the load messages not just the message that loads the type one node six part. If the neural net structure is generated randomly this could result in invalid outputs not being inhibited.

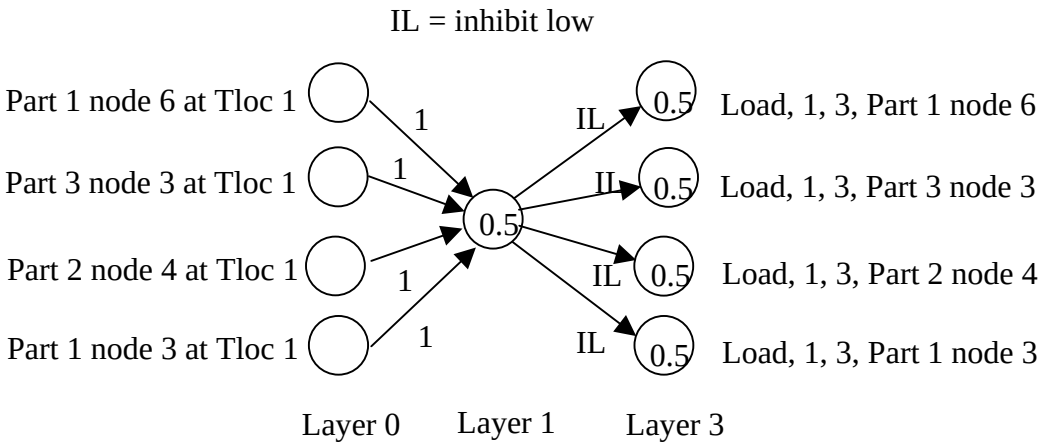

Figure 14 An Example of a Weak Rule

Additional nodes were added to identify conditions that would used frequently. These conditions were: a transporter at tlocation *x* has type 3 capacity, a transporter at tlocation *x* has type 4 capacity, a transporter is at tlocation *x*, a part is at tlocation *x*, workstation *j* needs type 3 capacity, workstation *j* needs type 4 capacity, workstation *j* is blocked by empty transporters, buffer *k* is empty.

Elements were then added to cause the nearest transporter with capacity to move to a workstation when the workstation needs capacity node was active, transporters that were closer but did not have capacity to move, and transporters that were past the workstation to move farther away until the transporter with capacity could reach the workstation that needed the capacity. If there was no transport capacity in the system and a processing workstation needed capacity then the transporter that was closest to a storage workstation load point was moved to the load point and a part removed from it.

5.6.2.4 Neural net logic construction data creation

Neural net logic construction data consists of a set of neural net inputs with a corresponding set of outputs. An input output set pairing is called an exemplar. The training data that is used in this research consists of single input-single output data. A single neuron in the input layer is paired with a single neuron in the output layer. When the input layer neuron is the only neuron that is on, it should cause the output layer neuron paired with it and no others to be on. The data set is created in a multi-step process. The first step is to identify all of the possible paths through the process plans from raw material to finished product. All paths begin at process plan node one (raw material) and end at node two (finished product). The paths were implemented as lists. All of the arcs leaving the start node were selected from the process plan arcs table. For each arc, a list was created with two nodes, the start node (list head) and the node at the head of the arc leaving the start node (list tail). Each list was then extended by adding nodes to the tail. The current tail was used to select arcs from the process plan, if there was only one arc, then the node at the head of the arc was added to the list as the new tail. If there was more than one arc, then the list was duplicated (number of arcs minus one copies were made) and each list (original plus copies) received a new tail. This process was repeated until each path terminated at the finished product node.

After all of the process plan paths have been identified, they are then converted to one or more equipment-based paths. The equipment-based path identifies the part type and part process node, the physical location of the part, the type of location, and the command that should be executed at that

physical location. There are potentially multiple equipment-based paths for each process plan path. Multiple equipment paths are created when there are movement options. Options are created when the workcell has multiple storage workstations, a storage workstation has multiple load or unload points, a processing workstation the part must visit has multiple load or unload points, or a processing workstation the part must visit contains a buffer.

After the equipment-based paths are completed, the corresponding neural net nodes are identified. To identify the input layer node, the status matrix row and column are first identified and then the input layer node that corresponds to that status matrix entry is identified. The status matrix row is identified using the part location and location type. The status matrix column is identified using the part type and process node. The output layer node is identified by finding the node that has a message matching the command that needs to be executed. After all of the paths have been processed, duplicate entries (entries with the same input and output values) are removed. Duplicate entries are created when there are multiple equipment paths for a single process plan path or when multiple process plan paths have a step or steps in common. Any time there are multiple process plan paths, the loading of the finished product into storage will generate duplicate entries.

The following example has been extracted from test case one. For details of test case one, see section 6. The process plan can be seen in Figure 5a. Table 16 shows the process plan paths for parts of type one. There are two paths because there are two alternative process to create the part. Table 17 shows the equipment path for process plan path number two from Table 16. The “START” command assigns the raw material its first processing step, node 4 in the process plan. The “UNLOAD” command initiates the removal of the part from storage (fixed part location one) to the unload point (transporter location two). The “LOAD” command causes the part to be taken from transporter location two (the load point for the processing workstation) and placed in fixed part location two (the material processing machine). The first process command causes the instructions for process node 4 to be executed and the second one causes the instructions for process node 5 to be executed. The

"UNLOAD" command then causes the part to be removed from fixed part location two to transporter location one (the unload point). It is not necessary for the neural net to specify the part type to be unloaded because the fixed part location can only have one part. The finished product is then loaded into the storage workstation with the "LOAD" command.

Table 16 Sample Process Plan Paths

PartNumber	PathNumber	Path
1	1	Part # 1, Path 1, 3, 2
1	2	Part # 1, Path 1, 4, 5, 2

Table 17 Sample Equipment Path

Path Number	Path Step	Part Type	Process Node	Location Identifier	Location Is FPL	Command
2	1	1	1	1	-1	START, PLOC= 1, TYPE= 1, ONODE= 1, NNODE= 4
2	2	1	4	1	-1	UNLOAD,1,2, TYPE= 1, NODE= 4
2	3	1	4	2	0	LOAD,2,2, TYPE= 1, NODE= 4
2	4	1	4	2	-1	PROCESS, PLOC= 2
2	5	1	5	2	-1	PROCESS, PLOC= 2
2	6	1	2	2	-1	UNLOAD,2,1
2	7	1	2	1	0	LOAD,1,1, TYPE= 1, NODE= 2

Table 18, Table 19, and Table 20 show the neural net data for the sample equipment path. Exemplar number 12 was deleted in the duplicate removal process because it was a duplicate of exemplar 5.

Table 18 Exemplar Identification Sample Equipment Path

Number	InputSize	OutputSize	EpathNumber
7	1	1	2
8	1	1	2
9	1	1	2
10	1	1	2
11	1	1	2
12	1	1	2
13	1	1	2

Table 19 Exemplar Input Values Sample Equipment Path

Number	NNNode	SMRow	SMCol	Value
7	46	2	2	0.95
8	49	2	5	0.95
9	29	1	5	0.95
10	69	3	5	0.95
11	70	3	6	0.95
12	67	3	3	0.95
13	7	0	3	0.95

Table 20 Exemplar Output Values Sample Equipment Path

Number	NNNode	Value	Message
7	686	0.95	START, PLOC= 1, TYPE= 1, ONODE= 1, NNODE= 4
8	660	0.95	UNLOAD,1,2, TYPE= 1, NODE= 4
9	608	0.95	LOAD,2,2, TYPE= 1, NODE= 4
10	604	0.95	PROCESS, PLOC= 2
11	604	0.95	PROCESS, PLOC= 2
12	626	0.95	UNLOAD,2,1
13	628	0.95	LOAD,1,1, TYPE= 1, NODE= 2

5.6.2.5 Neural net logic construction

The exemplars are used as the basis for creating the initial neural net structure for generating positive outputs. All previous neural net structure other than the initial node construction has been aimed at preventing an invalid output from being generated. A key to generating a successful control structure is that a single input should produce a single output. When there are alternatives (multiple paths through a process plan, or multiple load or unload points for a workstation) the exemplar creation process will generate multiple outputs for a single input. The neural net creation process must take this into account and create a structure that will only activate one of the outputs.

Initially, duplicate input values were located and the associated output nodes prioritized by the length of the path containing the output node. Prioritization was established by using a choice point (see Figure 10) where the threshold values indicated the priority. This priority scheme will generate an optimal flow time schedule for a single machine. For workcells with multiple machines, the schedule

will normally not be optimal since this prioritization scheme will only accept one path. Once a routing is selected, all parts will follow that route and the alternative routings will never be used.

The current system still uses the shortest processing time for prioritizing the process plan path that should be selected when starting a part (this is an area for future improvement) but does not use the prioritization scheme for other conflicts. Instead, the two choices are allowed to conflict so that an inhibit choice point is created when the controller is executed.

The next step is to add elements to prevent multiple messages from being sent to the same decision input place. Multiple messages are associated with storage workstations (there may be multiple parts ready to unload) and transporter capacities greater than one (multiple parts may want to load into a workstation or move the transporter).

5.6.2.6 Adaptation to deadlock situations

When a deadlocked or stalled condition occurs, a decision needs to be made by the control system regarding what new action should be taken. The Petri net is examined to determine if there is a transition that is *decision input fireable*. To be decision input fireable, a transition must have a decision input place in its pre-set and all of the places in its pre-set, except the decision input place, must be appropriately marked and all of the places in its post-set must have appropriate space. Figure 15 shows a decision input fireable transition taken from a Petri net sequence representing a transfer between material processors (from material processor two to material processor one). The shaded places to the left of the transition are marked and the unshaded places to the right of the transition are empty. If the decision input place was marked, the transition would fire. There should always be at least one decision input fireable transition in the Petri net, because the number of transporters in the system is required to be at least one less than the number of transporter locations so there will always be a transporter that can be moved.

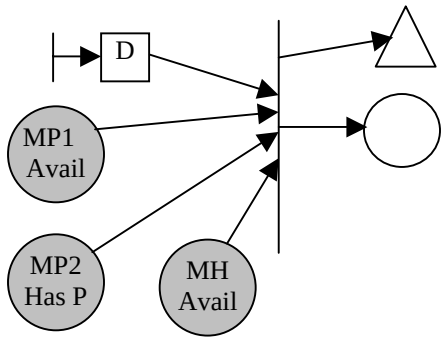

Figure 15 Decision Input Fireable Transition

After determining that at least one decision input fireable transition exists, the highest priority deadlock or stall condition is identified. Priorities are assigned based on part locations. From highest to lowest the priorities are: a part in a processing workstation, a part in the transportation system and an uncompleted part in a storage workstation. A list of parts, part carriers and transporters is developed by finding all of the non-zero entries in the status matrix. Multiple entries are added to the list for status matrix entries greater than one. This list is then separated into four categories. One category, parts in storage that should be in storage (i.e. raw material and finished product) is discarded and new separate lists are generated for the other three categories: parts in processing workstations, parts on transporters, uncompleted parts in storage.

The highest priority deadlock or stall type was then classified based on these new lists. The list of parts in processing workstations is processed first. The first entry on the list is used to select the processing workstation to have corrective measures applied to it. This gives priority to the workstation that contains the lowest numbered status matrix row because of the way the list is generated. A new list containing only the parts located in the workstation being corrected is then created. The workstation is then checked for deadlock.

If there is only one part in the workstation, then it can not be deadlocked and the problem is the part is blocked from unloading by a lack of transporter capacity. If there are two parts in the workstation,

then the current location of each part is compared with the desired location of the other part. If both parts are occupying the desired location of the other part then a circular wait condition exists and the type of deadlock is classified based on whether a buffer is present in the workstation. If a circular wait condition did not exist then the problem is that one of the parts wants to leave the workstation and can not because of lack of transporter capacity.

If there are three or more parts in the workstation then a multi-terminal shortest chain problem is created. An $n \times n$ matrix $\boldsymbol{A} = A_{ij}$ is used where n is the number of parts in the workstation. The rows and columns represented locations occupied by a part. A_{ij} is assigned a value of one if the part at location i wants to move to location j and a value of infinity otherwise. The distance from a node to itself, the A_{ii} diagonal, is infinity instead of the zero normally found in shortest chain problems. The problem was then solved using the procedure in Phillips and Garcia-Diaz (1981). After the problem is solved the values of the diagonal are checked. A value less than infinity indicates that the part at that location is involved in a circular wait. When a circular wait exists, the diagonal value also indicates the number of parts involved in the circular wait. Note that it is possible for large workstations (four or more machines) to have multiple circular wait conditions. The largest non-infinity value is selected as the circular wait condition to correct. The path matrix is then used to find the locations of the parts involved in the circular wait.

If a circular wait condition exists, then the deadlock is classified based on whether there is an available buffer in the workstation, whether there were buffers in the workstation that are occupied and if so whether all of the parts in the buffers want to remain in the workstation or whether at least one of the parts wants to leave the workstation. If no circular wait condition exists, then the problem is that at least one of the parts is blocked from unloading by lack of transporter capacity.

If there are no parts in a processing workstation, then the list of parts on transporters is checked. If parts exist then either a part has been moved off of the normal paths contained in the neural net logic

construction data or the transporter it was on is blocked from reaching the load point the part wants by an empty transporter.

If no parts are in processing workstations or on transporters then there must be uncompleted parts in a storage workstation either blocked because of lack of transport capacity or lack of controller logic. The neural net logic construction data assumes that once a part leaves the storage workstation it will be completed before reentering a storage workstation. This occurs because the neural net logic construction data is generated for a single part moving through the system.

After the type of the deadlock or stall is determined, elements are added to the neural net to initiate a recovery action. This involves generating a level 1 node to identify the deadlock or stall condition. Where there were multiple options to overcome the deadlock, the indicator node is linked to a choice point. For example, in a circular wait condition any of the two or more parts involved can be moved to an available buffer or, if there is no available buffer, unloaded. For conditions involving lack of transport capacity, the indicator node is linked to a level 2 node that indicates the workstation requires transport capacity of a particular type.

The introduction of inhibit choice points (ICPs) made it possible to create a control logic stall. This occurred when multiple conflicts had occurred and the conflicts involved subsets of the original conflict condition. Consider the case where three parts are available for unloading from storage. Part A is chosen to be unloaded leaving parts B and C in storage. Because the inhibit choice points are created based on pairs of incompatible outputs two ICPs were created where A was chosen over B and A was chosen over C. After A is unloaded an additional ICP will be created to determine whether B or C should be unloaded. Assume B is chosen over C. During performance tuning it will be possible for B to be chosen over A, A over C, and C over B resulting in all of the unload operations being inhibited. While it would be possible to add additional logic to cause one of the parts to be unloaded because all inhibition occurs at level 3 not at the preliminary output layer (layer 4), it was decided to terminate the simulation and move to the next genome in the performance tuning process.

5.6.2.7 Deadlock avoidance and/or prevention versus deadlock recovery

One of the basic assumptions of this research has been that a part can be unloaded from any machine in a processing workstation and removed from the workstation. Further, after the part has been removed from the workstation it can be reloaded into the workstation for further processing. Because of these assumptions and the hierarchical nature of the control system it is possible to "preempt" a part in a workstation, i.e. that is force it to give up the workstation resources it holds. The part no longer has control of when the workstation resources are released removing the third condition of Coffman *et al.* (1971) for deadlock. Preemption of some type is the basic deadlock recovery technique. When a circular wait condition is detected, one of the parts is removed from its current location forcing a release of the resources it holds. Generally special deadlock resolution resources must be available to allocate temporarily to the part that was forced to release resources, i.e. a deadlock recovery buffer. In this research, the transportation system and the storage workstation(s) that originally held the raw material are used instead of a dedicated deadlock recovery buffer. Further, it is assumed that the storage system has the capacity to store all raw material, in-process and finished parts. The number of parts in the system will therefore always be less than the storage capacity plus the transporter capacity so based on Co and Wysk (1986) the system will never reach a point where it cannot be undeadlocked.

Viswanadham *et al*. (1990) state, "Deadlocks usually arise as the final state of a complex sequence of operations on jobs flowing concurrently through the system and are thus generally difficult to predict." Deadlock detection is relatively easy compared to deadlock prediction. A general deadlock recovery mechanism can be developed where parts are transferred from the location they occupy when the deadlock is detected to a storage facility. At some point, the system must become undeadlocked and progress restart. The worst case scenario is that all but one part will have to be transported to a storage facility.

Allowing "deadlocks," that is, the creation of circular wait conditions, to occur may improve performance or harm performance depending on the configuration of the FMS. Consider an FMS

containing a processing workstation with two MP devices and a buffer. If the time to move between the MP device and the buffer is small compared to the processing time on the MP device then operating the MP devices in parallel is desirable even when it creates a circular wait between the two machines. If the transfer between machines and the processing time is small compared to the transfer time to the buffer then operating the machines in parallel may be undesirable when it causes a part to be transferred to the buffer.

One kind of deadlock that is always harmful is the situation where a material transport device is required to unload a machine and that material transport device has been assigned to another part that wants to use the machine that needs to be unloaded. Test case 2 has the potential for this type of deadlock. If a part is on the machine and a second part is unloaded from the storage workstation, the second part must be reloaded into the storage workstation before the part on the machine can be unloaded. To reduce this type of harmful delay, a deadlock reduction policy was implemented. The number of parts that could be unloaded from storage workstations was limited to the number of non-storage fixed part locations minus one plus the available transport capacity. For the single machine case, this guarantees that this type of deadlock will be avoided. For systems with more than one MP device, these deadlocks can still occur when the parts are released in an order that results in one of the MP devices being empty.

5.7 Genetic Algorithm Performance Tuning

The genome used for performance tuning is constructed with two strands of alleles. The two strands correspond to the set of choice points and the set of inhibit choice points. Each allele is the value of the choice that should be used for the choice point. The location of the allele (*locus*) is the position of the allele on the genome and corresponds to the choice or inhibit choice identifier. Three tables are used to store the genomes: GenomeID, GenomeChoicePointValues, and GenomeInhibitChoicePointValues. The GenomeID table has fields for the genome identifier, the performance value, the number of choice points and the number of inhibit choice points. As the

controller runs, the number of choice points and inhibit choice points may change. Recovering from a deadlock with a circular wait will add a choice point for the selection of the part to be removed from its current set of resources. Any set of choices that leads to resource contention will create an inhibit choice point. The GenomeChoicePointValues and GenomeInhibitChoicePointValues tables have fields for the genome identifier, the choice or inhibit choice point identifier, and the choice to be selected for the point.

To evaluate a genome the neural net is modified to match the choices specified by the genome. For each choice point the neuron associated with the selected choice has its threshold value set to the minimum threshold value specified for the choice point. The neurons associated with the non-selected choices have their thresholds set to the minimum threshold plus one. This threshold guarantees that the neurons will not be active, since it is greater than the possible sum of all of the inputs to the neuron. For each inhibit choice point, the arc associated with the selected choice has its weight value set to zero. The weights of the arcs associated with the non-selected choices are set to one. The arc with the weight of zero will have no effect on the node it would normally inhibit because the input value will be less than that required to inhibit the node. The controller is then operated in simulation mode and allowed to produce the batch of parts under consideration. When the part batch has been completed or the controller determines it cannot complete the batch, the objective function is computed and assigned to the genome as its performance value.

To achieve performance tuning a series of genomes are created and evaluated. The system used a steady-state population approach. After each genome was created, it was evaluated. If the genome performed better than the worst genome in the population then it replaced the worst genome in the population. Genomes were created and evaluated until the maximum number of simulations specified by the user was reached.

To create the initial population, a single genome was created where all of the choices and inhibit choices were assigned choice one. The genome was then evaluated. If the number of genomes was

less than one third of the maximum number to be kept in the population, the next genome was created by mutating the current best performer. Each locus was mutated with a probability of 0.35. If the locus was chosen for mutation, each possible allele, including the current one, was given equal probability of replacing the current allele. It was possible for an allele to replace itself. This effectively reduced the mutation rate. For a locus with two possible choices the probability that the locus changed was 0.175 (probability of selection 0.35 * probability of change if selected 0.5).

If the population size was between one third and two thirds of the maximum population size the next genome was created from two randomly selected parents. The alleles were selected from each parent with equal probability (i.e. probability of selecting the allele for locus *j* from parent A equals 0.5). When the population size was greater than two thirds of the maximum, the next genome was created from the best performer and a randomly selected member of the population. The best performer was modified by a crossover operator. The crossover operation could involve only the choice point strand, only the inhibit choice point strand, or both strands. The strands involved were selected randomly with equal weight given to the three options. The crossover operator functioned as both a two point crossover and a one point crossover. A starting point and an ending point for the crossover for each strand were randomly selected. If the value for the ending point was smaller than the value for the starting point, the operator acted as a one point operator, taking the strand from the starting point to the end of the strand. If the value for the ending point was larger, the operator took the strand from the starting point to the ending point.

5.8 Control System Operation

The operation of the controllers will now be described starting with the equipment level and working upward.

5.8.1 Equipment level

The operation of any individual equipment level controller is in general outside the scope of this research. The controller must respond to a set of standardized interface commands. The response to these commands will be machine specific. The controller will wait for a command to be received from

a higher level controller, convert that command to a machine specific command, perform the machine specific functions, monitor the process of the functions, and on completion of the functions send a message to the higher level controller indicating the command has been completed.

5.8.2 Workstation level

The processing workstation controllers are Petri nets that function as command expanders. A single activity sequence in the cell controller is expanded to a sequence of activities. Figure 16 shows the general operation of an event driven Petri net. There is an implicit assumption that there will not be any transitions that can fire before the first event happens. The workstation starts in the empty and idle condition and requires a load message before any transition will be able to fire. If the workstation controller is not starting from the empty and idle condition (i.e., it was stopped and then restarted) then the only transitions that will not have fired will be ones that are waiting for an event to occur. The other transitions are zero time events. While it is possible for the user to shut down the controller between zero time events, it is unlikely to happen.

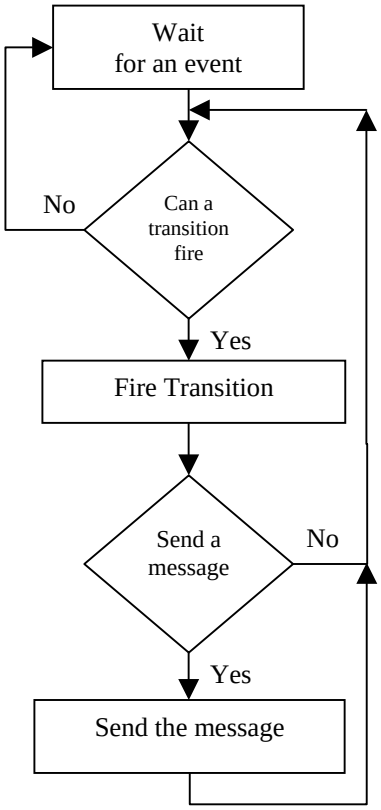

Figure 16 Workstation Controller Operation

Figure 17 shows a simple workstation controller in the empty and idle condition. The workstation has one material handler and one material processor with no buffers. The controller can perform three actions: load a part into the material processor, unload a part from the material processor, and perform a process on the processor. For each action, there is a corresponding decision input place, indicated in the figure by the square boxes containing the letter "D." The workstation initially contains two tokens, one in the place indicating the material processor is available, the other in the place indicating the material handler is available.

When a message is received from the cell controller, a token will be placed in the "Load Pending" decision input place (the associated event fired transition is not shown in Figure 17). After the "Load Pending" place receives a token the transition following it becomes enabled and fires, removing the

tokens from the three places in its pre-set and placing a token in each of the two places in its post-set. The tokens contain information about the type of part that should be loaded, and the transporter location and the mobile part location from which the part should be loaded. This information was contained in the message received from the cell controller.

The post-set contains a standard place used to indicate that an operation is in progress and an output place. The output place has a destination and a message format string associated with it that were assigned during the creation of the workstation controller. When the transition following the output place fires (it was enabled when the output place received a token), removing the token from the output place, a message is sent to the destination associated with the place using information held by the token and the output place format string. A token is also placed in the external clock place, indicating that a response is expected from another device. The transition between the external clock place and the input place is an event triggered transition. When the event occurs (the proper message is received from the controller associated with the event, in this case, a message indicating the Preparation has been completed), the transition fires placing a token in the input place. The process repeats itself as the transition starting the next activity (Picking) becomes enabled when the input place receives the token. The process of firing transitions and moving tokens continues until no transitions are enabled. This occurs after the transition following the “Load Complete” output place (not shown in Figure 17) fires. The Petri net will have tokens in the “material handler available” place and the “MP has a part” place. The system then waits until the next event occurs placing a token in one of the “decision input places.”

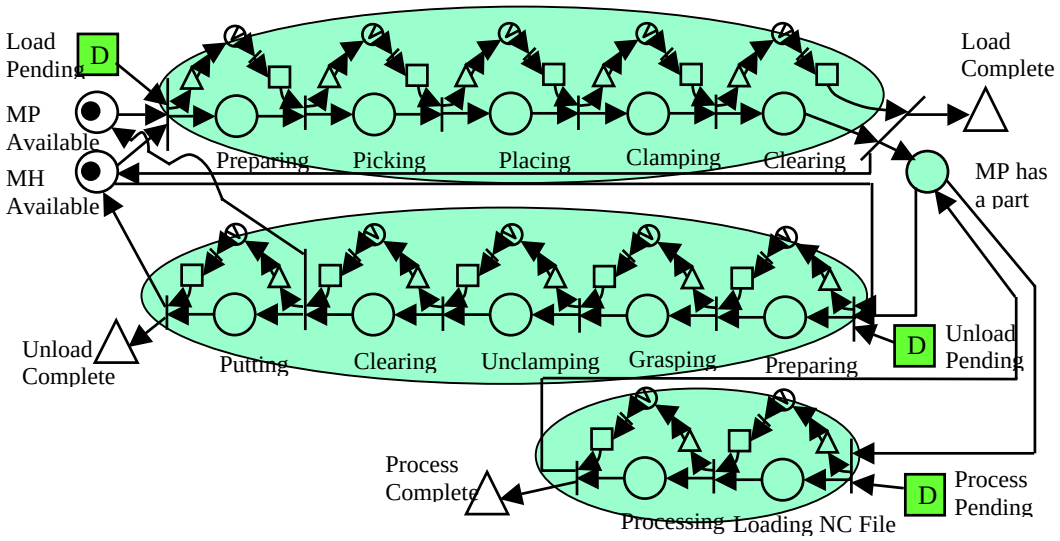

Figure 17 Simple Workstation Controller

5.8.3 Cell level

The cell controller initiates all actions in the workcell. The cell controller is a combination of a Petri net and a neural net. The workcell is represented as a Petri net. The Petri net performs the execution function while the neural net performs the decision making function. Figure 18 shows the general function of the controller. The Petri net attempts to fire each transition. If the transition fires, a flag is set to indicate the Petri net should be restarted after the neural net has been processed. As part of the transition firing process, the rows in the status matrix associated with the places in the transition's pre- and post-sets are updated.

After attempting to fire all Petri net transitions, the neural net is processed using the updated status matrix. To ensure that the neural net does not make decisions based on conditions that no longer exist, the Petri net is processed four times for each processing of the neural net. Four was used because the number of transitions associated with an activity is four, i.e. an initial transition fires, a message sending transition fires, a message received transition fires and then a final activity complete transition

fires. This is done to allow actions initiated by the previous neural net decisions to be reflected in the status matrix.

The results of the neural net processing are a set of messages that are sent to the Petri net. The set may be empty. After the neural net is processed, the flag indicating whether a Petri net transition fired during the last processing of the Petri net is checked. If a transition fired, then the Petri net is processed again. If a transition did not fire, then the Petri net is not processed, because no transition will be capable of firing. The system must wait until an event occurs. When the message arrives that triggers the event, the transition associated with the event is fired, the transition fired flag is set, and the Petri net is processed, restarting the Petri net -- Neural net processing cycle.

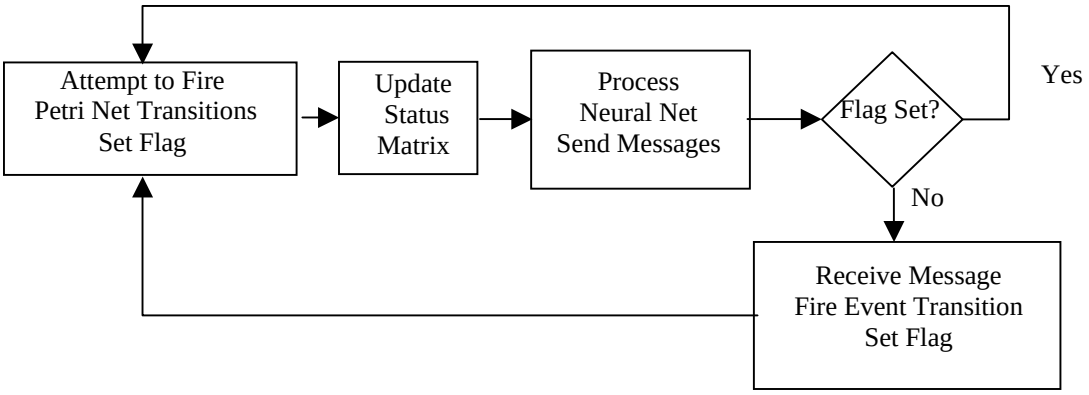

Figure 18 Cell Control Operation

If no Petri net transitions fire in the current cycle and the neural net does not initiate any decisions then the status matrix is checked to determine if a stalled / deadlocked condition is present. The rows that are marked for deadlock detection represent activities in progress. If these rows do not have any non-zero entries then there are no activities in progress so no future events will occur. The system is stalled or deadlocked and will remain in its current state indefinitely unless the deadlock recovery and adaption procedure discussed in section 5.6.2.6 is executed.

5.9 Control System Summary

To operate the cell controller, storage workstation and equipment level controllers that respond to the specified interface must be provided. Simple processing workstation controllers that act as command expanders can be created automatically from the user input model of the workcell. The interesting portion of the control system is the cell controller. The cell controller uses a Petri net to model the workcell and an artificial neural net to model the control logic. The control logic can be extracted into human understandable rules because the weights of the neural net have been restricted (with one exception) to integer values resulting in a Boolean logic. The exception was used to implement an "OR" operation and the logic remains Boolean. With an additional neural net layer the "OR" operation could be performed using integer value weights.

A genetic algorithm is used to performance tune the controller. The genome is used to select among choices of specific operations instead of the more common selection of a heuristic to make scheduling decisions.

The inputs to the choice and inhibit choice points were created using data limited to the workstation involved in the choice, so the controller is not using all of the information available to it. This means that a guaranteed global optimum will not be achieved. The current usage of the choice points, where only a single choice is allowed, is less efficient than the original idea of allowing additional connections to the choice point where the thresholds of the nodes are used to indicate priority or preference. This would allow the choice to change dynamically instead of being fixed. A method for selecting the additional connections and allowing the connections to be changed to tune the controller needs to be developed.

6 EXAMPLE IMPLEMENTATION

A description of the process of generating the control system for test case one will now be presented.

Figure 19 shows the user input process, the facility side of the figure will be presented first. Test case one is a very simple workcell consisting of a storage workstation, a processing workstation, and two transporter locations. Both transporter locations are occupied by a transporter loaded with a part carrier. Only parts are moved between workstations, not part carriers. Parts are required to be on part carriers while stored in the storage workstation or while on a transporter. They are not on a part carrier while in the processing workstation. Four types of parts are processed in the workstation. The scheduling objective is minimum average flow time. For a single machine system, the shortest processing time first (SPT) heuristic is known to optimize average flow time.

The first step is to identify the equipment and other resources (i. e. part carriers and transporters) used in the workcell. Table 21 lists the equipment used by test case one, while Table 22 shows the transporter types that are used and Table 23 shows the part carrier types. After the equipment and resources are identified, the various locations can be specified. There is a fixed part location for each piece of storage equipment and each material processor. Table 24 identifies the fixed part locations in test case one. The other fixed locations (the transporter stopping locations) are then specified. Table 25 shows the two transporter locations in test case one. The mobile part locations with their associated transporter type are then specified. Table 26 shows the single mobile part location associated with the single transporter type.

Table 21 Test Case One Equipment

EquipmentNumber	EquipmentDescription	EquipmentType	ControllerName
1	Automated Storage	AS	AS1
2	Storage Robot	MH	StorageMH
3	Material Processor	MP	MetalCutter
4	Processing Robot	MH	CuttingRobot
5	Transportation System	MT	BigMover

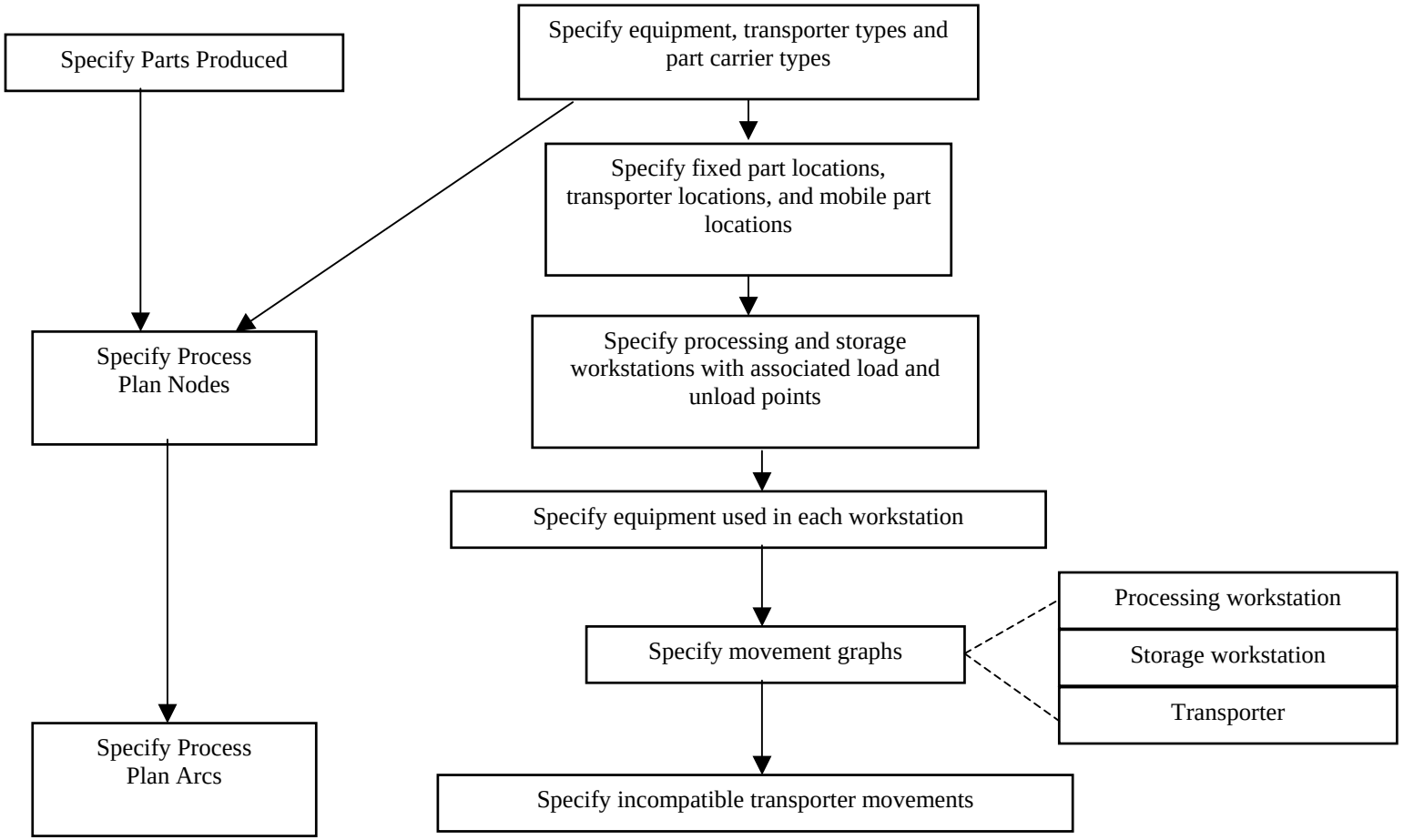

Figure 19 User Input Process

Table 22 Test Case One Transporter Types

TransporterTypeNumber	PlocationCount	TransporterDescription
1	1	The only type

Table 23 Test Case One Part Carrier Types

PartCarrierTypeNumber	PartCarrierDescription	TransporterType
1	Holds a single part	1

Table 24 Test Case One Fixed Part Locations

LocationNumber	LocationDescription	EquipmentNumber
1	Storage	1
2	Material Cutter	3

Table 25 Test Case One Transporter Locations

LocationNumber	LocationDescription	EquipmentNumber	LoadPoint	UnLoadPoint
1	Storage Load	5	Yes	Yes
2	Storage Unload	5	Yes	Yes

Table 26 Test Case One Mobile Part Locations

LocationNumber	Location Description	TransporterType
1	The one and only	1

After the physical system has been specified, the organization of the workcell is specified by identifying the processing (see Table 27) and storage (see Table 28) workstations with their associated load (see Table 29 and Table 31) and unload (see Table 30 and Table 32) points, then the equipment is assigned to the appropriate workstation (see Table 33 and Table 34). The transportation system is not assigned to a workstation because the cell controller directly controls the movement between workstations.

Table 27 Test Case One Processing Workstations

WorkstationNumber	Description	ControllerName
1	Processing Workstation	PWS1

Table 28 Test Case One Storage Workstations

WorkstationNumber	Description	ControllerName
1	Storage Workstation	SWS1

Table 29 Test Case One Processing Workstation Load Points

WorkstationNumber	TlocationNumber
1	2

Table 30 Test Case One Processing Workstation Unload Points

WorkstationNumber	TlocationNumber
1	1

Table 31 Test Case One Storage Workstation Load Points

WorkstationNumber	TlocationNumber
1	1

Table 32 Test Case One Storage Workstation Unload Points

WorkstationNumber	TlocationNumber
1	2

Table 33 Test Case One Processing Workstation Equipment

Workstation Number	Equipment Number
1	3
1	4

Table 34 Test Case One Storage Workstation Equipment

Workstation Number	Equipment Number
1	1
1	2

After the workstations have been defined and the equipment assigned, the movement possibilities must be defined. Three sets of graphs are created by defining arcs between the locations previously defined. The first graph describes how transporters move through the transportation system. Test case one does not allow transporters to move, so there are no arcs in this graph. The other two sets of graphs define how parts move in relationship to the workstations (see Table 35 and Table 36). As previously mentioned, workstation movement graphs use the three location data points (LocationData1, LocationData2, LocationData3) to store the end points of the arc in a from - to configuration. The meaning of the location data changes depending on the type of arc. Load arcs store their origin using LocationData1 to store the transporter location and LocationData2 to store the mobile part location. LocationData3 is used to store the fixed part location where the part should be placed after it is removed from the transporter. Unload arcs store their origin (a fixed part location) in LocationData1 and their destination in LocationData2 (transporter location) and LocationData3 (mobile part location). Transfer arcs store the origin fpl in LocationData1 and the destination fpl in LocationData2. Transfer arcs do not use LocationData3.

The final step in identifying the movement possibilities is to identify the transporter movements that are not compatible. If two arcs are incompatible, there will be two entries in the table. Test case one does not allow transporter movement, so it does not have any incompatible transporter movements.

Table 35 Test Case One Processing Workstation Movement Graph Arcs

Workstation Number	Arc Number	Equipment Number	Estimated Time	Type of Arc	Part Only	Location Data1	Location Data2	Location Data3
1	1	4	15	1	Yes	2	1	2
1	2	4	15	3	Yes	2	1	1

Table 36 Test Case One Storage Workstation Movement Graph Arcs

Workstation Number	Arc Number	Equipment Number	Estimated Time	Type of Arc	Part Only	Location Data1	Location Data2	Location Data3
1	1	2	30	1	Yes	1	1	1
1	2	2	30	3	Yes	1	2	1

At this point, the facility description has been completed. The parts that are to be processed in the system must now be specified. This specification begins with a list of the parts that are to be produced (see Table 37). After the parts are listed, all of the raw material, finished product and processing nodes in the part process plans are specified (see Table 38). Each processing node has associated with it a piece of equipment, an instruction set, and an estimated processing time. The estimated processing time is used to simulate the performance of the workcell during training. The process plan nodes are then connected together with process plan arcs to specify the manufacturing constraints (see Table 39). After the process plan arcs have been specified, the user input is complete.

Table 37 Test Case One Part Identification

PartNumber	PartName	PartDescription
1	One	Test case one has alternative route
2	Two	Single option one step
3	Three	Alternative Process allowed
4	Four	Alternative process shorter than single step

Table 38 Test Case One Process Plan Nodes

PartNumber	NodeNumber	Equipmentnumber	Instructions	EstimatedTime
1	1	1	Raw Material	0
1	2	1	Finished Product	0
1	3	3	NC-1-1	600
1	4	3	NC-1-2	500
1	5	3	NC-1-3	120
2	1	1	Raw Material	0
2	2	1	Finished Product	0
2	3	3	NC-2-1	400
3	1	1	Raw Material	0
3	2	1	Finished Product	0
3	3	3	NC-3-1	800
3	4	3	NC-3-2	450
3	5	3	NC-3-3	450
4	1	1	Raw Material	0
4	2	1	Finished Product	0
4	3	3	NC-4-1	750
4	4	3	NC-4-2	200
4	5	3	NC-4-3	475

Table 39 Test Case One Process Plan Arcs

ArcNumber	PartNumber	StartingNode	EndingNode
1	1	1	3
2	1	3	2
3	1	1	4
4	1	4	5
5	1	5	2
1	2	1	3
2	2	3	2
1	3	1	3
2	3	3	2
3	3	1	4
4	3	4	5
5	3	5	2
1	4	1	3
2	4	3	2
3	4	1	4
4	4	4	5
5	4	5	2

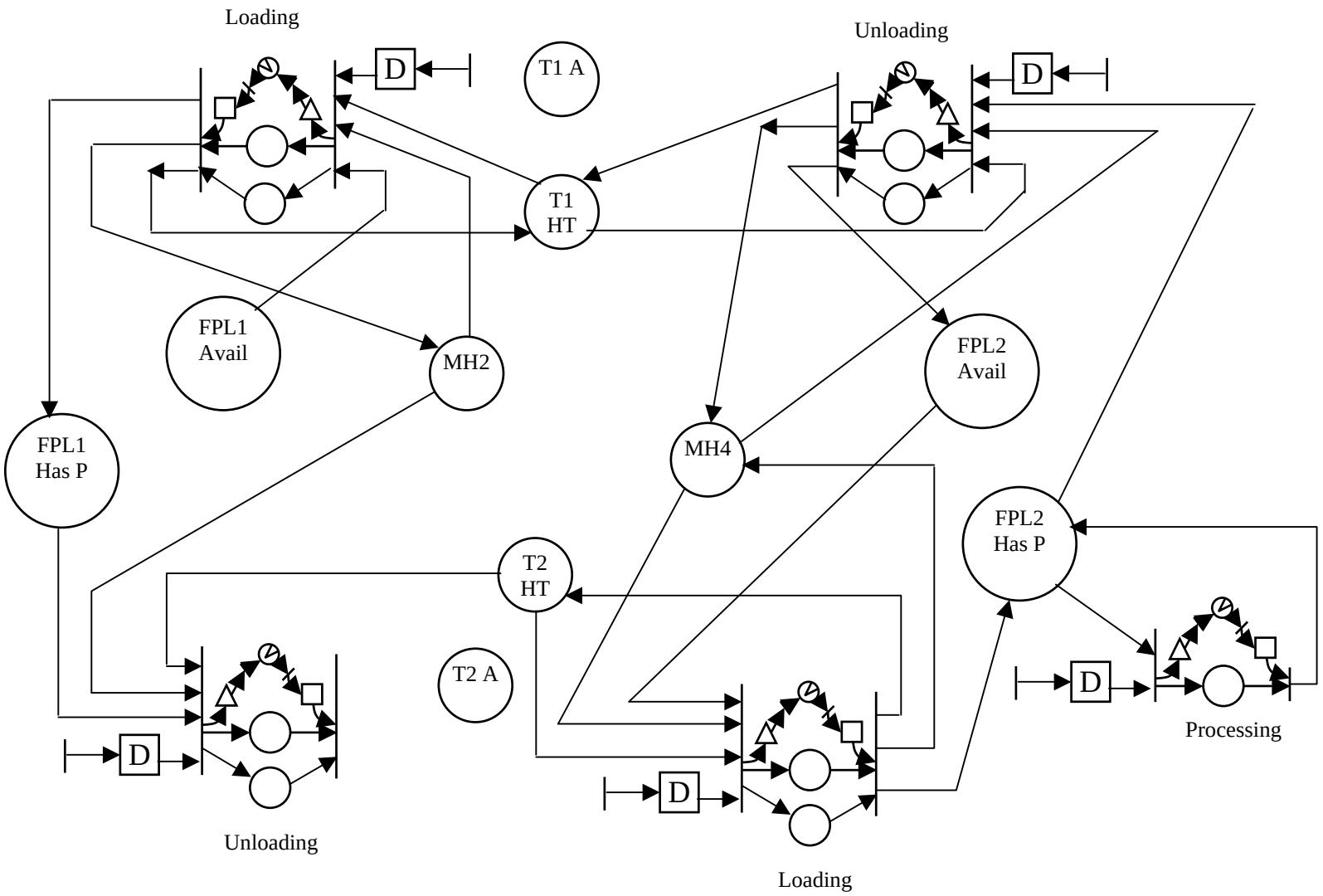

Figure 20 Test Case One Partial Cell Controller Petri Net

The user can now build the controllers. It does not matter whether the processing workstation controller or the cell controller is created first. Separate controller construction programs are used for the processing workstation and the cell controller. The processing workstation controller for test case one will look like the controller in Figure 17.

The workcell controller consists of two parts, the Petri net that interacts with the other controllers and the neural net that makes the decisions. The Petri net portion is partitioned into several distinct groupings. There is one grouping that represents the facility and one grouping for each part process plan. Figure 20 shows the unmarked Petri net grouping that represents the test case one facility.

The neural net was constructed with the minimum number of nodes necessary to represent the inputs, outputs and the control rules derived from the Petri net. The input layer consisted of 604 nodes. Four nodes represent the orders for the four part types. The other 600 nodes represent the status of the workcell organized as a thirty row, twenty column, status matrix. The rows represent physical locations and activities, Table 40 defines the meanings of the rows. The rows that are labeled as "Transforming" indicate that a new process plan node is being assigned to the part. The old node label is the node currently assigned to the part and the new node label is the one that will be assigned to the part after the transformation. The columns represent physical objects in the system (parts, part carriers and transporters), Table 41 defines the meanings of the columns. The output layer consisted of 47 nodes, each of these nodes has an associated message that represents a possible decision the neural net can reach. These messages are listed in Table 42 along with the preliminary output layer node associated with the message. The shortest path through each process plan was selected manually. Table 43 shows the shortest paths arranged in increasing length order.

Table 40 Test Case One Status Matrix Row Definitions

Status Matrix Row	Interpretation
0	Tlocation 1 Has a Transporter
1	Tlocation 2 Has a Transporter
2	FPL1 Has a Part
3	FPL2 Has a Part
4	Processing Ploc2
5	Loading Ploc2 from Tloc2 Transporter
6	Loading Ploc2 from Tloc2 Part
7	Unloading Ploc2 to Tloc1 Transporter
8	Unloading Ploc2 to Tloc1
9	Loading Ploc1 from Tloc1 Transporter
10	Loading Ploc1 from Tloc1 Part
11	Unloading Ploc1 to Tloc2 Transporter
12	Unloading Ploc1 to Tloc2
13	Transforming Ploc1, Type1, Old node 1, New node 3
14	Transforming Ploc1, Type1, Old node 1, New node 4
15	Transforming Ploc2, Type1, Old node 3, New node 2
16	Transforming Ploc2, Type1, Old node 4, New node 5
17	Transforming Ploc2, Type1, Old node 5, New node 2
18	Transforming Ploc1, Type2, Old node 1, New node 3
19	Transforming Ploc2, Type2, Old node 3, New node 2
20	Transforming Ploc1, Type3, Old node 1, New node 3
21	Transforming Ploc1, Type3, Old node 1, New node 4
22	Transforming Ploc2, Type3, Old node 3, New node 2
23	Transforming Ploc2, Type3, Old node 4, New node 5
24	Transforming Ploc2, Type3, Old node 5, New node 2
25	Transforming Ploc1, Type4, Old node 1, New node 3
26	Transforming Ploc1, Type4, Old node 1, New node 4
27	Transforming Ploc2, Type4, Old node 3, New node 2
28	Transforming Ploc2, Type4, Old node 4, New node 5
29	Transforming Ploc2, Type4, Old node 5, New node 2

Table 41 Test Case One Status Matrix Columns Definitions

Status Matrix Column	Interpretation
0	Transporter Type 1
1	Part Carrier Type 1
2	Part Type 1 Node 1 (Raw material)
3	Part Type 1 Node 2 (Finished Product)
4	Part Type 1 Node 3
5	Part Type 1 Node 4
6	Part Type 1 Node 5
7	Part Type 2 Node 1 (Raw material)
8	Part Type 2 Node 2 (Finished Product)
9	Part Type 2 Node 3
10	Part Type 3 Node 1 (Raw material)
11	Part Type 3 Node 2 (Finished Product)
12	Part Type 3 Node 3
13	Part Type 3 Node 4
14	Part Type 3 Node 5
15	Part Type 4 Node 1 (Raw material)
16	Part Type 4 Node 2 (Finished Product)
17	Part Type 4 Node 3
18	Part Type 4 Node 4
19	Part Type 4 Node 5

Table 42 Test Case One Neural Net Output Messages

IDNumber	Message
604	PROCESS, PLOC= 2
606	LOAD,2,2, TYPE= 1, NODE= 3
608	LOAD,2,2, TYPE= 1, NODE= 4
610	LOAD,2,2, TYPE= 1, NODE= 5
612	LOAD,2,2, TYPE= 2, NODE= 3
614	LOAD,2,2, TYPE= 3, NODE= 3
616	LOAD,2,2, TYPE= 3, NODE= 4
618	LOAD,2,2, TYPE= 3, NODE= 5
620	LOAD,2,2, TYPE= 4, NODE= 3
622	LOAD,2,2, TYPE= 4, NODE= 4
624	LOAD,2,2, TYPE= 4, NODE= 5
626	UNLOAD,2,1
628	LOAD,1,1, TYPE= 1, NODE= 2
630	LOAD,1,1, TYPE= 1, NODE= 3
632	LOAD,1,1, TYPE= 1, NODE= 4
634	LOAD,1,1, TYPE= 1, NODE= 5
636	LOAD,1,1, TYPE= 2, NODE= 2
638	LOAD,1,1, TYPE= 2, NODE= 3
640	LOAD,1,1, TYPE= 3, NODE= 2
642	LOAD,1,1, TYPE= 3, NODE= 3
644	LOAD,1,1, TYPE= 3, NODE= 4
646	LOAD,1,1, TYPE= 3, NODE= 5
648	LOAD,1,1, TYPE= 4, NODE= 2
650	LOAD,1,1, TYPE= 4, NODE= 3
652	LOAD,1,1, TYPE= 4, NODE= 4
654	LOAD,1,1, TYPE= 4, NODE= 5
656	UNLOAD,1,2, TYPE= 1, NODE= 2
658	UNLOAD,1,2, TYPE= 1, NODE= 3
660	UNLOAD,1,2, TYPE= 1, NODE= 4
662	UNLOAD,1,2, TYPE= 1, NODE= 5
664	UNLOAD,1,2, TYPE= 2, NODE= 2
666	UNLOAD,1,2, TYPE= 2, NODE= 3
668	UNLOAD,1,2, TYPE= 3, NODE= 2
670	UNLOAD,1,2, TYPE= 3, NODE= 3
672	UNLOAD,1,2, TYPE= 3, NODE= 4
674	UNLOAD,1,2, TYPE= 3, NODE= 5
676	UNLOAD,1,2, TYPE= 4, NODE= 2
678	UNLOAD,1,2, TYPE= 4, NODE= 3
680	UNLOAD,1,2, TYPE= 4, NODE= 4
682	UNLOAD,1,2, TYPE= 4, NODE= 5
684	START, PLOC= 1, TYPE= 1, ONODE= 1, NNODE= 3
686	START, PLOC= 1, TYPE= 1, ONODE= 1, NNODE= 4
688	START, PLOC= 1, TYPE= 2, ONODE= 1, NNODE= 3
690	START, PLOC= 1, TYPE= 3, ONODE= 1, NNODE= 3
692	START, PLOC= 1, TYPE= 3, ONODE= 1, NNODE= 4
694	START, PLOC= 1, TYPE= 4, ONODE= 1, NNODE= 3
696	START, PLOC= 1, TYPE= 4, ONODE= 1, NNODE= 4

Table 43 Test Case One Shortest Process Plan Paths

Part	Process Plan Path	Processing Time
2	1→3→2	400
1	1→3→2	600
4	1→4→5→2	200 + 475 = 675
3	1→3→2	800

The Petri net decision input places were used to create control rules (see Table 44, Table 45 and Table 46). If adding a token to a decision input place would not enable the transition following the decision input place then decisions that would place a token in the decision input place should not be made. The control rules used nodes 698 to 713 to represent conditions that need to be met to allow a decision to be valid. The duplicate nodes in Table 44 are generated when the part process plan has alternative routes. Each route has a decision input node associated with choosing that route. The conditions for the feasibility of routes are identical, raw material must be available and the number of parts that have been started must be less than the number of parts that have been ordered.

After the cell controller was built, the neural net was adjusted manually to implement the SPT heuristic. This adjustment consisted of changing the arc type of the link from the conditions to the start output node from 2 (inhibit low) to 3 (excitatory fixed weight) see Table 46 and adding additional nodes (see Table 47) and arcs (see Table 48, Table 49, and Table 50). Movement activities were prioritized from highest to lowest: Load PWS, Unload PWS, Unload SWS, Load SWS. See Appendix F for a detailed description of the logic development.

Table 44 Test Case One Control Rule Conditions

Node	Layer	Interpretation
698	1	Fpl 2 has a part
699	1	Tloc 2 has a transporter and a part, the material handler is not active, and there is no part processing or idle at FPL 1
700	1	There is a part idle at FPL 1
701	1	Tloc 1 has a transporter with space available to accept a part
702	2	There is a part idle at FPL 1, Tloc 1 has space available to accept a part, and the material handler is not active
703	1	Tloc 1 has transporter and part, material handler is not active
704	1	FPL 1 has a part
705	1	Tloc 2 has a transporter with space available to accept a part
706	2	FPL 1 has a part, Tloc 2 has space for a part, and the material handler is not active
707	1	Raw material for part 1 is available and the number of type ones that have been started is less than the number of type 1 parts ordered
708	1	Duplicate of 707
709	1	Raw material for part 2 is available, and the number of type 2 parts started is less than the number ordered
710	1	Raw material for part 3 is available, and the number of type 3 parts started is less than the number ordered
711	1	Duplicate of 710
712	1	Raw material for part 4 is available, and the number of type 4 parts started is less than the number ordered
713	1	Duplicate of 712

Table 45 Test Case One Petri Net Control Rules

Condition Node (s)	Link Type	Output Node (s)	Interpretation
698	2	604	A part can not be processed if it is not in the processing workstation
699	2	606,608,610,612,614, 616,618,620,622,624	The processing workstation can not be loaded if it already has a part, there is not part available, or the material handler is busy
702	2	626	The processing workstation can not be unloaded if it does not have a part, there is not space at the unload point, or the material handler is busy
703	2	628,630,632,634,636, 638,640,642,644,646, 648,650,652,654	A part can not be placed in the storage workstation unless there is a part at the load point and the material handler is not busy
706	2	656,658,660,662,664, 666,668,670,672,674, 676,678,680,682	A part can not be taken from the storage workstation unless there is a part in the workstation, the material handler is not busy, and there is space available at the unload point

Table 46 Test Case One Modified Petri Net Rules

Condition Node (s)	Link Type	Output Node (s)	Interpretation
707	3	684	Conditions are correct to start part type 1
707	1	690,696	Don't start part type 3 or 4 if a type 1 part can be started
708	2	686	Don't start part 1 alternate path unless raw material is available and fewer parts are started than ordered
709	3	688	Conditions are correct to start part type 2
709	1	684,690,696	Don't start part type1, 3 or 4 if a type 2 part can be started
710	3	690	Conditions are correct to start part type 3
711	2	692	Don't start part 3 alternate path unless raw material is available and fewer parts are started than ordered
712	2	694	Don't start part 4 path unless raw material is available and fewer parts are started than ordered
713	3	696	Conditions are correct to start part type 4
713	1	690	Don't start part type 3 if a type 4 part can be started

Table 47 Test Case One Manually Added Nodes

Node	Layer	Interpretation
714	1	Part type 2 has been ordered
715	1	Part type 1 has been ordered
716	1	Part type 4 has been ordered
717	1	Part type 3 has been ordered
718	1	A part is ready to be processed (P1N3, P2N3, P3N3, P4N4, P4N5)
719	1	A completed part is ready to be unloaded (P1N2, P2N2, P3N2, P4N2)
720	1	P2N3 is in the storage workstation
721	1	P1N3 is in the storage workstation
722	1	P4N5 is in the storage workstation
723	1	P4N4 is in the storage workstation
724	1	P3N3 is in the storage workstation
725	1	Tloc 2 has P2N3 (storage unload point, processing load point)
726	1	Tloc 2 has P4N5 (storage unload point, processing load point)
727	1	Tloc 2 has P1N3 (storage unload point, processing load point)
728	1	Tloc 2 has P4N4 (storage unload point, processing load point)
729	1	Tloc 2 has P3N3 (storage unload point, processing load point)
730	1	Tloc 1 has P2N2 (storage load point, processing unload point)
731	1	Tloc 1 has P1N2 (storage load point, processing unload point)
732	1	Tloc 1 has P4N2 (storage load point, processing unload point)
733	1	Tloc 1 has P3N2 (storage load point, processing unload point)

Table 48 Test Case One Manually Added Arcs from Manually Added Nodes

Input NodeID	OutPut NodeID	Weight	LinkType	Interpretation
718	604	1	3	A part is ready to be processed so process it
719	626	1	3	A completed part is ready to unload so unload it
720	658	-1	3	P2N3 is in storage do not unload P1N3
720	666	1	3	P2N3 is in storage unload it
720	670	-1	3	P2N3 is in storage do not unload P3N3
720	680	-1	3	P2N3 is in storage do not unload P4N4
720	682	-1	3	P2N3 is in storage do not unload P4N5
721	658	1	3	P1N3 is in storage unload it
721	670	-1	3	P1N3 is in storage do not unload P3N3
721	680	-1	3	P1N3 is in storage do not unload P4N4
722	658	-1	3	P4N5 is in storage do not unload P1N3
722	670	-1	3	P4N5 is in storage do not unload P3N3
722	680	-1	3	P4N5 is in storage do not unload P4N4
722	682	1	3	P4N5 is in storage unload it
723	670	-1	3	P4N4 is in storage do not unload P3N3
723	680	1	3	P4N4 is in storage unload it
724	670	1	3	P3N3 is in storage unload it
725	612	1	3	Tloc 2 has P2N3 load it into the PWS
726	624	1	3	Tloc 2 has P4N5 load it into the PWS
727	606	1	3	Tloc 2 has P1N3 load it into the PWS
728	622	1	3	Tloc 2 has P4N4 load it into the PWS
729	614	1	3	Tloc 2 has P3N3 load it into the PWS
730	636	1	3	Tloc 1 has P2N2 load it into the SWS
731	628	1	3	Tloc 1 has P1N2 load it into the SWS
732	648	1	3	Tloc 1 has P4N2 load it into the SWS
733	640	1	3	Tloc 1 has P3N2 load it into the SWS

Table 49 Test Case One Arcs Manually Added to Manually Added Nodes

InputNodeID	OutPutNodeID	Weight	LinkType	Interpretation
1	714	1	3	P2 ordered
0	715	1	3	P1 ordered
3	716	1	3	P4 ordered
2	717	1	3	P3 ordered
68	718	1	3	P1N3 in PWS
73	718	1	3	P2N3 in PWS
76	718	1	3	P3N3 in PWS
82	718	1	3	P4N4 in PWS
83	718	1	3	P5N5 in PWS
67	719	1	3	P1N2 in PWS
72	719	1	3	P2N2 in PWS
75	719	1	3	P3N2 in PWS
80	719	1	3	P4N2 in PWS
53	720	1	3	P2N3 in SWS
48	721	1	3	P1N3 in SWS
63	722	1	3	P4N5 in SWS
62	723	1	3	P4N4 in SWS
56	724	1	3	P3N3 in SWS
33	725	1	3	P2N3 in Tloc 2
43	726	1	3	P4N5 in Tloc 2
28	727	1	3	P1N3 in Tloc 2
42	728	1	3	P4N4 in Tloc 2
36	729	1	3	P3N3 in Tloc 2
12	730	1	3	P2N2 in Tloc 1
7	731	1	3	P1N2 in Tloc 1
20	732	1	3	P4N2 in Tloc 1
15	733	1	3	P3N2 in Tloc 1

Table 50 Test Case One Manually Added Arcs to Generated Nodes

InputNodeID	OutPutNodeID	Weight	LinkType	Interpretation	
709	684	1	1	P2 can start so don't start P1 (N3)	
707	690	1	1	P1 can start so don't start P3 (N3)	
709	690	1	1	P2 can start so don't start P3 (N3)	
713	690	1	1	P4 can start so don't start P3 (N3)	
707	696	1	1	P1 can start so don't start P4 (N4)	
709	696	1	1	P2 can start so don't start P4 (N4)	
86	699	1	1	P1N1 is processing	So Tloc 2 has a transporter and a part, the material handler is not active, and there is no part processing or idle at FPL 1 is false
87	699	1	1	P1N2 is processing	
88	699	1	1	P1N3 is processing	
89	699	1	1	P1N4 is processing	
90	699	1	1	P1N5 is processing	
91	699	1	1	P2N1 is processing	
92	699	1	1	P2N2 is processing	
93	699	1	1	P2N3 is processing	
94	699	1	1	P3N1 is processing	
95	699	1	1	P3N2 is processing	
96	699	1	1	P3N3 is processing	
97	699	1	1	P3N4 is processing	
98	699	1	1	P3N5 is processing	
99	699	1	1	P4N1 is processing	
100	699	1	1	P4N2 is processing	
101	699	1	1	P4N3 is processing	
102	699	1	1	P4N4 is processing	
103	699	1	1	P4N5 is processing	

7 TESTING PROCEDURE AND RESULTS

The control system was tested using four cases. Test case one was used primarily for debugging the program code and as a simple demonstration of the controller concept. Test case two was used to test stall recovery and cycle avoidance. Test case 3 expanded the transportation system and added a second machine to the processing workstation creating the possibility of a circular wait within the processing workstation. Test case 4 added a buffer to the processing workstation. All possible deadlock types were available using the four test cases.

A weighted flowtime with all time categories weighted equally was used as the objective function. This is functionally equivalent to the sum of the completion times for the parts where completion is defined as a completed part being placed in storage. All load, unload, transfer and transporter movement times were assumed to be independent of the part or transporter type. Load and unload operations to storage workstations were assigned a duration of 30. Load and unload operations to processing workstations were assigned a duration of 15. Transfer operations within processing workstations were assigned a duration of 20.

The original neural network design called for a fully-connected three-layer network with real weights. The quantity of data required to determine appropriate weights was intractable. The network was then changed to a three-layer network with integral weights where the links could be constructed to represent Boolean logic. Test case one demonstrated that a three layer network did not have enough depth. A layer was added to allow generation of conditions of the form: *IF cond1 and cond2 and not cond3 and not cond4 THEN output x is ON*. Conditions of this type were required when generating the neural net rules associated with Petri net decision input places (see section 5.6.2.3). Test case two and the work done to implement deadlock and stall recovery showed that the four layer network that worked for test case one was inadequate and an additional layer was added. Further deadlock recovery development required an “ORing” of conditions that required an additional layer be added to the

network to maintain integral weights. An exception to the integral weight rule was made and the five layer network was found to be sufficient to construct all of the required logic.

7.1 Test Case One

Tokens representing raw material for two parts of each part type with associated part carriers were added to the empty and idle Petri net marking generated by the controller creation program. An order for one part of each part type was placed in the order vector. The cell controller was then operated in simulation mode to generate performance data.

The control logic for test case one was developed three times. The first set of control logic was developed manually and implemented the shortest processing time first heuristic, which is known to be optimal for the single-machine scheduling problem. This logic was discussed in section 6. The second set of control logic was developed automatically with an early version of the building programs using the neural net example data. This version used fixed priority paths. The paths were prioritized with the shortest path having highest priority. The size of the controllers generated by these two methods is shown in Table 51. The third version was developed with the final building program that used choice and inhibit choice points enabling the paths priority to be changed.

Test case one demonstrated that the controller concept was viable. The Petri net did maintain the state information describing the workcell. The neural net with appropriately designed weights functioned as a set of logic rules that implemented the shortest processing time first algorithm. Because of the construction of the workcell, it was impossible to generate a deadlock situation, so the deadlock detection mechanism of the controller was not tested.

Test case one is single machine scheduling problem where shortest processing time first is known to be optimal for minimizing mean flow time. To achieve this the parts should be processed in the order: 2, 1, 4, 3, following the routing shown in Table 52. The activities that minimize flowtime are shown in

Table 53. The sum of the completion times is 6090 with the parts completing at the times listed in Table 54.

Table 51 Test Case One Logic Comparison

Element	Manual Logic	Automated Logic
Layer 0 nodes	604	604
Layer 1 nodes	34	50
Layer 2 nodes	2	41
Layer 3 nodes	47	47
Layer 4 nodes	47	47
Links	1277	1376

Table 52 Flowtime Minimizing Part Processing Sequence

Part Identifier	Processing Steps
2	3 (400)
1	3 (600)
4	4 (200), 5 (475)
3	3 (800)

Table 53 Flowtime Minimizing Activities

Activity	Start Time	Finish Time
Unload part 2 from storage	0	30
Load part 2 to machine	30	45
Unload part 1 from storage	30	60
Process part 2 node 3	45	445
Unload part 2 from machine	445	460
Load part 1 to machine	460	475
Load part 2 to storage	460	490
Process part 1 node 3	475	1075
Unload part 4 from storage	490	520
Unload part 1 from machine	1075	1090
Load part 4 to machine	1090	1105
Load part 1 to storage	1090	1120
Process part 4 node 4	1105	1305
Unload part 3 from storage	1120	1150
Process part 4 node 5	1305	1780
Unload part 4 from machine	1780	1795
Load part 3 to machine	1795	1810
Load part 4 to storage	1795	1825
Process part 3	1810	2610
Unload part 3 from machine	2610	2625
Load part 3 to storage	2625	2655

Table 54 Optimal Part Completion Times

Part Identifier	Finish time
2	490
1	1120
4	1825
3	2655

An initial test of twenty schedules was run with the mutation rate at 0.35. The best result found from 20 schedules was 6390 achieved by 4 different schedules (genomes 2, 4, 6, 7). All four schedules selected the part routes found in Table 55. Table 56 shows the activity sequence generated and Table 57 shows the part completion times. The degradation of 300 is caused by two things: part 1 was processed before part 2 (200) and the longer processing path was selected for part 3 (100). The genomes were compared and found to be almost identical. The choice strand was identical for all four genomes. The inhibit point strands were different lengths; however, the addition of loci past the minimum length strand will have no effect if the first portion of the strand is identical to the shortest length strand. The choices represented by those loci after the minimum length strand represent choices that will not be required. Two of the extended strands were identical to the shortest length strand. The third differed at only one locus. This uniformity indicates the mutation rate was two low in the genome creation process. The mutation rate was increased to 0.85. The best result found from 20 genomes (maximum population 30) was 6470. The best result found from 500 genomes (maximum population 50) was 6240. Two hundred and forty-one of the genomes reached the 6240 result.

Table 55 Genome Part Processing Path Selection

Part	Process Step
1	3
2	3
3	4, 5
4	4, 5

Table 56 Generated Activity Sequence with Best Flowtime

Activity	Start Time	Finish Time
Unload part 1 node 3 from storage	0	30
Load part 1 to machine	30	45
Process part 1	45	645
Unload part 2 node 3 from storage	45	75
Unload part 1 from machine	645	660
Load part 2 to machine	660	675
Load part 1 to storage	660	690
Process part 2	675	1075
Unload part 4 node 4 from storage	690	720
Unload part 2 from machine	1075	1090
Load part 4 to machine	1090	1105
Load part 2 to storage	1090	1120
Process part 4	1105	1305
Unload part 3 from storage	1120	1150
Process part 4 second step	1305	1780
Unload part 4 from machine	1780	1795
Load part 3 to machine	1795	1810
Load part 4 to storage	1795	1825
Process part 3	1810	2260
Process part 3 second step	2260	2710
Unload part 3	2710	2725
Load part 3 to storage	2725	2755

Table 57 Generated Part Completion Times

Part Identifier	Finish time
1	690
2	1120
4	1825
3	2755

7.2 Test Case Two

Test case two was a modified version of test case one. One of the transporters was removed and two arcs were added to the transporter movement graph. In test case one, it is optimal to place a part from the storage workstation on the transporter occupying the processing workstation load point as soon as the transporter is empty (the part has been loaded into the processing workstation). In test case two, the same action will generate a deadlocked condition because the transporter must be moved to the processing workstation unload point to allow the part to be removed from the processing workstation. Test case two is very similar to the simple manufacturing system analyzed by Viswanadham *et al.* (1990) with the transporter serving the function of the AGV.

To clear the deadlock, the transporter will be moved to the storage workstation load point and the part on the transporter placed in the storage workstation. The result is that an extra unload operation (causing the deadlock) and an extra load operation (to clear the deadlock) are executed whenever there is more than one part in the storage workstation ready to be processed. The extra unload operation will not delay the processing of any parts because it occurs simultaneously with the processing of the part ahead of it. It will result in extra movement of the material handler possibly resulting in additional maintenance requirements. The extra load operation does delay the processing of parts because the material processor is blocked for the length of time required to reload the part into the storage workstation. The material processor cannot be unloaded until the storage workstation load is completed.

A simple deadlock prevention policy can be implemented for this system. The deadlock is created when the storage workstation is unloaded filling the space that is required to unload the processing workstation. By inhibiting storage workstation unload operations when there is a part in the processing workstation, it would always be possible to unload the processing workstation and deadlocks would not occur. More generally if the number of parts in the transport system plus the number of parts in non-storage fixed part locations is one less than the capacity of the transport system plus the capacity of the fixed part locations then deadlock will not occur for a system organized like test case 2. This rule can be applied to any system but will not prevent deadlocks in all systems. Consider a system with the same transportation system as test case two and where the processing workstation has two unique machines A and B that are not interchangeable. If parts requiring the same machine are released sequentially then the system will deadlock. The part on the machine will not be able to unload because the transporter is occupied and the part on the transporter cannot be loaded into the processing workstation because the machine is occupied. The number of parts in the system will be less than the capacity because of the empty space on the unused machine so the storage workstation unload will not be prevented. For test case two the rule is a deadlock prevention policy, but in the

more general case it is a deadlock reduction policy. It reduces the number of deadlock states that can be reached but does not eliminate them.

This is the same single-machine scheduling problem with delays introduced by the unavailability of transportation capacity. The flow time minimizing part sequence is still that of test case one: 2, 1, 4, 3. The storage workstation unload operation can no longer overlap the processing operations because the transporter that would be filled by the storage unload is required to unload the processing workstation. Also, due to the implementation of the control system, the processing workstation will not request that the transporter move to the unload point until the part has completed processing, this causes an extra delay in the unload operation that could be eliminated. Table 58 shows the activities, which now include transporter movements. The part finish times for the non-concurrent moves show cumulative delays. The first part is delayed 20, the second 40, the third 60 and the fourth 80. Table 59 shows the optimal part completion times. The sum of completion times is 6570 for the concurrent move case and 6770 for the non-concurrent move case.

Three hundred control choice sets were generated; one hundred forty-three of them found the non-concurrent move optimum performance value of 6770. The result was first found with control set 45. Table 60 shows the messages generated by the neural net. Transformation messages generated by the Petri net portion of the controller were included in Table 60 to show the completion of part processing.

Table 58 Optimal Flow Time Activities

	With concurrent moves		Without concurrent moves	
Activity	Start Time	Finish Time	Start Time	Finish Time
Unload part 2 from storage	0	30	0	30
Load part 2 to machine	30	45	30	45
Process part 2 node 3	45	445	45	445
Move transporter	45	65	445	465
Unload part 2 from machine	445	460	465	480
Load part 2 to storage	460	490	480	510
Move Transporter	490	510	510	530
Unload part 1 from storage	510	540	530	560
Load part 1 to machine	540	555	560	575
Process part 1 node 3	555	1155	575	1175
Move Transporter	555	575	1175	1195
Unload part 1	1155	1170	1195	1210
Load part 1 to storage	1170	1200	1210	1240
Move transporter	1200	1220	1240	1260
Unload part 4 from storage	1220	1250	1260	1290
Load part 4 to machine	1250	1265	1290	1305
Process part 4 node 4	1265	1465	1305	1505
Move transporter	1265	1285		
Process part 4 node 5	1465	1940	1505	1980
Move transporter			1980	2000
Unload part 4 from machine	1940	1955	2000	2015
Load part 4 to storage	1955	1985	2015	2045
Move transporter	1985	2005	2045	2065
Unload part 3 from storage	2005	2035	2065	2095
Load part 3 to machine	2035	2050	2095	2110
Process part 3 node 3	2050	2850	2110	2910
Move transporter	2050	2070	2910	2930
Unload part 3	2850	2865	2930	2945
Load part 3 to storage	2865	2895	2945	2975

Table 59 Optimal Part Completion Times with Transporter Movements

	Finish times	
Part Identifier	Concurrent moves	Without concurrent moves
2	490	510
1	1200	1240
4	1985	2045
3	2895	2975

Table 60 Test Case 2 Neural Net Messages

Source	Time	Message
Neural	0	START, PLOC= 1, TYPE= 1, ONODE= 1, NNODE= 3 START, PLOC= 1, TYPE= 2, ONODE= 1, NNODE= 3 START, PLOC= 1, TYPE= 3, ONODE= 1, NNODE= 3 START, PLOC= 1, TYPE= 4, ONODE= 1, NNODE= 4
	0	UNLOAD,1,2, TYPE= 2, NODE= 3
	30	LOAD,2,2, TYPE= 2, NODE= 3
	45	PROCESS, PLOC= 2
Petri	445	BigExec:PWS1:TRANSFORM, PLOC= 2, TYPE= 2, ONODE= 3, NNODE= 2
Neural	445	MOVE,2,1
	465	UNLOAD,2,1
	480	LOAD,1,1, TYPE= 2, NODE= 2
	510	MOVE,1,2
	530	UNLOAD,1,2, TYPE= 1, NODE= 3
	560	LOAD,2,2, TYPE= 1, NODE= 3
	575	PROCESS, PLOC= 2
Petri	1175	BigExec:PWS1:TRANSFORM, PLOC= 2, TYPE= 1, ONODE= 3, NNODE= 2
Neural	1175	MOVE,2,1
	1195	UNLOAD,2,1
	1210	LOAD,1,1, TYPE= 1, NODE= 2
	1240	MOVE,1,2
	1260	UNLOAD,1,2, TYPE= 4, NODE= 4
	1290	LOAD,2,2, TYPE= 4, NODE= 4
	1305	PROCESS, PLOC= 2
Petri	1505	BigExec:PWS1:TRANSFORM, PLOC= 2, TYPE= 4, ONODE= 4, NNODE= 5
Neural	1505	PROCESS, PLOC= 2
Petri	1980	BigExec:PWS1:TRANSFORM, PLOC= 2, TYPE= 4, ONODE= 5, NNODE= 2
Neural	1980	MOVE,2,1
	2000	UNLOAD,2,1
	2015	LOAD,1,1, TYPE= 4, NODE= 2
	2045	MOVE,1,2
	2065	UNLOAD,1,2, TYPE= 3, NODE= 3
	2095	LOAD,2,2, TYPE= 3, NODE= 3
	2110	PROCESS, PLOC= 2
Petri	2910	BigExec:PWS1:TRANSFORM, PLOC= 2, TYPE= 3, ONODE= 3, NNODE= 2
Neural	2910	MOVE,2,1
	2930	UNLOAD,2,1
	2945	LOAD,1,1, TYPE= 3, NODE= 2
Finish	2975	Genome 167 is among the 50 best with a score of 6770

7.3 Test Cases Three and Four

Test case three was an expansion of test case two. A second machine was added to the processing workstation and a third transporter location was added. Test case four was created by adding a buffer to test case three. The configuration of test case three is shown in Figure 21. The four parts used in

test cases one and two were used and a fifth part added that needed to be processed on the new machine. The fifth part had two processing alternatives: 1) process for 800 time units on the new machine, 2) process for 450 time units on the new machine followed by 450 time units on the old machine. These test cases provided for the possibility of concurrent processing on the two machines. There was also the possibility of a circular wait condition where the part on the transporter wanted a machine and a part on the machine wanted to unload requiring the transporter. Because there are two machines the deadlock reduction policy is not a deadlock prevention policy as discussed in section 7.2.

The system created one of these circular wait conditions and began to correct it. The transporter was moved away from the processing load point to the storage workstation load point. The system then created an inhibit choice point between removing the part from the transporter and moving the transporter to location 2. The choice to move the transporter was selected and the system entered a state of continuous cycling around the transportation system.

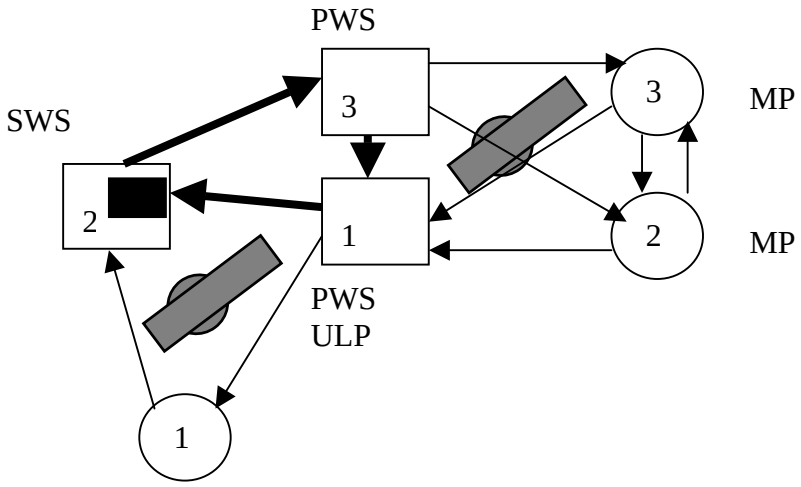

Figure 21 Test Case 3 Configuration

8 CONTRIBUTIONS, SUGGESTED FUTURE RESEARCH, AND CONCLUSIONS

Flexible manufacturing research has been subject to the "Blind men and the Elephant" problem where individual researchers have been developing control system pieces in isolation with only a limited shared vision of what the control system should look like when complete. Further, researchers tend to publish only general concepts and not specific implementation details. The lack of implementation details means replicating a specific piece of work, if possible, requires significant effort that goes unrewarded.

One result of this lack of detail is that there is no agreement on the details and function of an equipment level controller. If there is one thing that is generally agreed upon, it is that an equipment level controller is required. Naylor and Volz (1987) even discussed the structure of such a controller, partitioning it into two parts: one to deal with hardware specific issues related to the equipment it was controlling, the other to provide a standard interface to the rest of the control system. Yet, no standard equipment controller exists. One of the advantages of a standardized equipment controller would be the ability to easily share the flexible manufacturing systems held at various research facilities. Some of this advantage could be achieved without standardization, by publishing the details of the control interfaces of the various systems. This would be a significant boon to small research groups that cannot afford the cost of maintaining their own FMS or to groups that are just entering the research field.

8.1 Contributions

This research was designed to develop a cell level controller that could be rapidly generated for a hierarchical control system. Three basic hypotheses were to be verified: 1) a factory reference model could be developed to a level where implementation was unambiguous, 2) a Petri net model could be generated from the factory reference model (objective three had been partially accomplished in that

Petri nets had been selected as the modeling system when this hypothesis was generated), and 3) an artificial neural net could be generated given the factory reference model and the Petri net that would generate valid control actions that would result in factory performance with some degree of "goodness." Five objectives were defined as steps to demonstrate the hypotheses above (see section 3.2).

The first two objectives were used to demonstrate that hypothesis one was true. There was actually no doubt that it was true, given that other researchers have previously developed control systems. Any control system that is developed has an implied reference model. What was needed was to document a reference model to the point where it could be easily replicated. Objective one was to create a specification for equipment controller interfaces. To complete this objective, a set of equipment level activities was defined (see Table 2). A set of standard messages was then developed to signal the initiation and completion of these activities (see Table 5 and Table 7). Objective two was to create a specification allowing an FMS user to input a description of the FMS and a database to hold the information. A model consisting of a set of process plan graphs and two sets of movement graphs was designed. A set of database tables was specified (see Table 4 and Appendix B) and implemented using Microsoft Access. A small FMS example was used to describe the data required (see section 6).

Objective three was to develop a model for the workcell and parts and to be able to automatically generate this model from the data entered by the user. Modified Petri nets were selected to model the workcell and the part process plans. Petri nets have been previously used to model process plans and workcells individually. Because Petri nets are a form of graph, constructing Petri nets from the graphs used to model the FMS was relatively straight-forward. The graphs used to in the FMS representation used directed arcs to model activities and nodes to represent locations or part conditions. A Petri net structure that corresponded to an activity was created. Preconditions and post-conditions were developed based on the activity represented by the model arc and the nodes it connected. The

translation algorithm is contained in Appendix H. See Appendix C for a description of the output database that holds the resulting Petri net.

Hypothesis three was originally stated as two hypotheses: 1) a neural net could be generated and 2) scheduling knowledge could be induced in the neural net. Objectives four and five are based on the original hypotheses. Objective four was the creation of a basic neural net structure. Objective five was adjusting the basic structure to generate "good" results. The neural net structure was developed using the Petri net model to specify the size of the input and output layers. The input layer consisted of a node for each element of the status matrix and order vector. The status matrix was a construct developed to convert information contained in the marking of the Petri net into a usable form. The order vector was a user input indicating how many parts were to be manufactured. The output layer consisted of a node for each decision (control action) the controller was required to express (initiate). A partial control logic was created based on the Petri net decision input places. The manual development of a logic for test case one (see Appendix F) demonstrated that the neural net structure could be used as a controller.

Objective five, automatically creating scheduling knowledge was a more difficult task to meet than demonstrating that the neural net controller concept was usable. The only system states where known correct (optimal) control actions could be found were those states entered when processing a single part in the workcell. The positive elements were extracted from these states (i.e. all state variables that were zero were ignored) and used to create a control logic. Because the control logic was created ignoring state variables that were zero, it allowed conflicting actions to be generated. A method of identifying these conflicts and selecting among the conflicting actions (inhibit choice points) was developed. A genetic algorithm was then used to process the sets of choices that developed as the system evolved.

8.2 Suggested Future Research

Additional activities could be added to the ones proposed. The most significant of these would be a refixturing activity where a part is removed from a material-processing device by a material handler and then replaced in the same material processor in a different orientation. This will be a relatively common requirement when multiple reference surfaces must be created on a part. Another activity that would be useful is a transporter transfer operation where a part is removed from one transporter and placed on another transporter. This would allow modeling systems with multiple transportation systems (such as a conveyor and an AGV system) something that is not currently allowed (the transporter movement graph was assumed to be a strongly connected digraph).

The transporter movement rules could be revised. The exemplar-based neural net generation originally created exemplars to move transporters from workstation load points to unload points. This was deactivated because it moved any transporter (including ones containing parts that needed to load into the workstation) not just empty transporters. The model translation program was then later modified (during development devoted to deadlock recovery) to add neural net nodes that indicated if a transportation location was occupied (had a transporter) and whether a part was located at the transportation location. With these nodes available, it should be possible to add logic to move only empty transporters from load points to unload points. As was shown in Table 58, making the movement of the transporters concurrent with the part processing results in a better schedule.

While controller size is not likely to be a significant limitation for the workcells considered, the number of neural net inputs could be reduced by changing the way transformation tracking is handled. There is currently a status matrix row for each part transformation that occurs (i.e. each process plan arc). All entries in this row must be zero except the one column that represents the part entering the transformation, so only one neural net input is needed per row. Because transformations are zero time events and the controller processes the Petri net four times for each time the neural net is processed, it would be possible to completely eliminate the transformation indicator rows when operating in

simulation mode. The rows will always be zero when the neural net is processed. In operational mode, the rows are not guaranteed to be zero because events can occur on any part of the Petri net processing cycle.

A logic translation program that converted the neural net structure to human understandable Boolean logic rules and back would be a useful tool. Having the rules in Boolean logic form would allow the users to develop insight into the operation of the workcell and allow for tuning or pruning of the logic rules by a control expert. Also, additional rules could be added to the controller building logic. Potential rules include:

1. a CONWIP style limit on the number of parts in the workcell
2. limiting the number of parts in the system that have been assigned to a workstation
3. limiting the number of parts in the system assigned to any machine

Even more rules could be added if time based inputs were added. The e-clock places in the Petri net have the potential to indicate the time remaining before their associated processes are completed. A system clock would have to be added to the controller and some method of providing the information to the neural net. Perhaps by augmenting the status matrix with an additional column, since all e-clock places have an associated activity that has a specific row in the status matrix.

While the system was developed as a controller not a scheduler, job shop scheduling may be possible using the model shown in Figure 22. By setting all material handling times to zero and placing a transporter in the tlocation with a capacity equal to or greater than the number of jobs to be scheduled, the system represents the normal assumptions made when doing job shop scheduling.

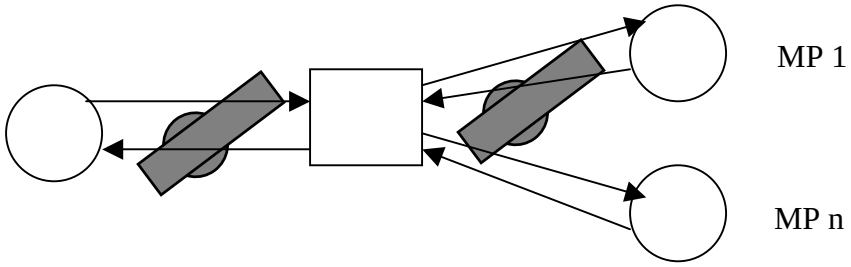

Figure 22 Job Shop Representation Model

8.3 Conclusions

The proposed control system structure is viable although additional development is necessary. The separation of the control logic (the artificial neural net) from the model of the manufacturing cell (the Petri net) makes automatic generation of the controller possible. It will also make alternative approaches to building control logic (such as a fuzzy neural net) easy to implement. The equipment controller interface specification and the detailed user input model will allow other researchers to easily apply the controller to their manufacturing systems.

Because the controller is automatically generated from an easy to construct / modify model of the workcell, it has a very high degree of "expansion flexibility." This expansion flexibility makes the controller appropriate for small manufacturers that are implementing their first FMS.

REFERENCES

Adlemo, A., Andreasson, S.-A., Fabian, M., Gullander. P., and Lannartsson, B. (1995) Towards a Truly Flexible Manufacturing System. *Control Eng. Practice*, **3(**4), pp. 545-554.

Ang W.L. and Bundell, G.A. (1996) A Petri Net Based Task Scheduler as a Real-Time FMS Controller, in *Proceedings of the 1996 IEEE Conference on Emerging Technologies and Factory Automation*, Hawaii, pp 738-744, Nov 18-21.

Banaszak, Z. A. and Krogh, B. H. (1990) Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows. *IEEE Transactions on Robotics and Automation*, **6**(6), pp. 724-734.

Bose, N.K. and Liang, P. (1996) *Neural Network Fundamentals with Graphs, Algorithms, and Application.*, McGraw-Hill, Inc., New York

Chan, C.-C. and Wang, H.-P. (1993) Design and Development of a Stochastic High-Level Petri Net System for FMS Performance Evaluation, *International Journal of Production Research*, **31(**10), pp. 2415-2439.

Chittipeddi, K. and Wallet, T. (1991) Entrepreneurship and Competitive Strategy for the 1990s. *Journal of Small Business Management*, Jan, pp. 94-98.

Cho, H. and Wysk, R. A., (1995) An Intelligent Workstation Controller for Computer Integrated Manufacturing: Problems and Models, *Journal of Manufacturing Systems*, **14**(4), pp. 252-263.

Chryssolouris, G. and Lee, M. (1992) An Assessment of Flexibility in Manufacturing Systems. *Manufacturing Review*, **5**(2), pp. 105-116.

Co, C.H. and Wysk, R.A. (1986) The Robustness of CAN-Q in Modeling Automated Manufacturing Systems. *International Journal of Production Research*, **27(**6), pp. 1485-1503.

Coffman, E.G. Jr., Elphick, M., and Shoshani, A. (1971) System Deadlocks. *Comput. Surveys*, **3**(2), pp. 67-78, June.

Drake, G. (1996) A Framework for On-line Simulation Systems. Master Thesis, Texas A&M University, College Station.

Drake, G., Smith, J.S., and Peters, B.A. (1995) Simulation as a Planning and Scheduling Tool for FMS, in *Proceedings of the 1995 Winter Simulation Conference*, December, Washington, D.C., pp. 805-812.

Duffie, N.A., Chitturi, R. and Mou, J. (1988) Fault-tolerant Heterarchical Control of Heterogeneous Manufacturing System Entities. *Journal of Manufacturing Systems*, **7**(4), pp. 315-327.

Evert, B. (1980) *Cluster Analysis*. Heinemann, New York.

Ezpeleta, J. and Colom, J. M. (1997) Automatic Synthesis of Colored Petri Nets for the Control of FMS. *IEEE Transactions on Robotics and Automation*, **13**(3), pp. 327-337.

Ezpeleta, J., Colom, J. M. and Martinez, J. (1995) A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems. *IEEE Transactions on Robotics and Automation*, **11**(2), pp. 173-184.

Florin, G. and Natkin, S. (1982) Evaluation Based upon Stochastic Petri Nets of the Maximum Throughput of a Full Duplex Protocol, In *Application and Theory of Petri Nets*, C.Girault and W. Reisig (eds), Springer, New York, pp. 280-288.

Foo, Y.P.S. and Takefuji, Y. (1988) Integer Linear Programming Neural Networks for Job-shop Scheduling, in *Proc. 1988 International IEEE Conference Neural Networks*, **2**, pp. 341-348.

Gowan, J. A. and Mathieu, R. G. (1996) Critical Factors in Information System Development for a Flexible Manufacturing System, *Computers in Industry*, **28**, pp. 173-183.

Gupta, M. and Cawthon, G. (1996) Managerial Implications of Flexible Manufacturing for Small/Medium-sized Enterprises. Technovation, **16**(2), pp. 77-83.

Haddock, J. and O'Keefe, R. M. (1990) Using Artificial Intelligence to Facilitate Manufacturing Systems Simulation. *Computers and Industrial Engineering*, **18**(3), pp. 275-283.

Hemant Kumar, N. S. and Srinivasan, G. (1996) A Genetic Algorithm for Job-shop Scheduling -- A Case Study. *Computers in Industry*, **31**, pp. 155-160.

Herrman, J. W., Lee, C. and Hinchman, J. (1995) Global Job Shop Scheduling with a Genetic Algorithm. *Production and Operations Management*, **4**(1), pp. 30-45.

Holland, J.H. (1975) *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

Hwang, J.-L., and Henderson, M.R. (1992) Applying the Perceptron to Three-Dimensional Feature Recognition. *Journal of Design and Manufacturing*, **2**(4), pp. 187-198.

Jensen, K. (1992) *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Springer, New York.

Jensen, K. and Rozenberg, G., (eds) (1991) *High-Level Petri Nets: Theory and Application*, Springer-Verlag, New York.

Jones, A.T. and McLean, C.R. (1986) A Proposed Hierarchical Control Model for Automated Manufacturing Systems. *Journal of Manufacturing Systems*, **5**(1), pp. 15-25.

Jones, A.T and Saleh, A. (1990) A Multi-Level / Multi-Layer Architecture for Intelligent Shopfloor Control. *International Journal of Computer Integrated Manufacturing*, **3**(1), pp. 60-70.

Kamarthi, S.V., Kunara, S.T., Yu, F.T.S. and Ham, I. (1990) Neural Networks and Their Applications in Component Design Data Retrieval. *Journal of Intelligent Manufacturing*, **1**(2), pp. 125-140.

Kelton, W.D., Sadowski, R.P. and Sadowski, D.A. (1997) *Simulation with Arena*, McGraw Hill, New York.

Kempenaers, J, Pinte, J., Detand, J. and Kruth, J.-P. (1996) A Collaborative Process Planning and Scheduling System. *Advances in Engineering Software*, **25**, pp. 3-8.

Knapp, G.M. and Wang, H.-P. (1992a) Neural Networks in Acquisition of Manufacturing Knowledge, in Kusiak, A. (ed), *Intelligent Design and Manufacturing*, , John Wiley & Sons, New York, pp. 723-744.

Knapp, G.M. and Wang, H.-P. (1992b) Acquiring, Storing, and Utilizing Process Planning Knowledge Using Neural Networks. *Journal of Intelligent Manufacturing*, **3**(5), pp. 333-344.

Kumara, S.R.T. and Ham, I. (1990) Use of Associative Memory and Self-Organization in Conceptual Design. *Annals of the CIRP*, **39**(1), pp. 117-120.

Kumara, S.R.T. and Kamarthi, S.V. (1991) Function-to-Structure Transformation in Conceptual Design: An Associative Memory-Based Paradigm. *Journal of Intelligent Manufacturing*, **2**(5), pp. 281-292.

Kumaran, T.K., Chang,W., Cho, H. and Wysk, R.A. (1994) A Structured Approach to Deadlock Detection, Avoidance and Resolution in Flexible Manufacturing Systems. *International Journal of Production Research*, **32**(10), pp. 2361-2379.

Law, A.M. and Kelton, W.D. (1991) *Simulation Modeling and Analysis*, McGraw Hill, New York.

Lawley, M., Reveliotis, S., and Ferreira, P. (1997) Design Guidelines for Deadlock Handling Strategies in Flexible Manufacturing Systems. *International Journal of Flexible Manufacturing Systems*, **9**(1), January, pp. 5-30.

Lee, A. (1994) *Knowledge-Based Flexible Manufacturing Systems (FMS) Scheduling*, Garland, New York.

Lee, D.Y., and DiCesare, F. (1994) Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search. *IEEE Transactions on Robotics and Automation*, **10**(2), pp. 123-132.

Leung, Y.T. and Sheen, G.-J. (1993) Resolving Deadlocks in Flexible Manufacturing Cells. *Journal of Manufacturing Systems*, **12**(4), December, pp. 291-304.

Li, D.C. and She, I. S. (1994) Using Unsupervised Learning Technologies to Induce Scheduling Knowledge for FMSs. *International Journal of Production Research*, **32**(9), pp. 2187-2199.

Liu, X. and Zhang, W.J. (1998) Issues on the Architecture of an Integrated General-Purpose ShopFloor Control Software System. *Journal of Materials Processing Technology*, **76**, pp. 261-269.

MacGregor, R.J. (1987) *Neural and Brain Modeling*, Academic Press, London.

McCulloch, W.S. and Pitts, W.A. (1943) A Logical Calculus of the Ideas Immanent in Neural Nets. *Bull. Math Biophysics*, **5**, pp. 115-133.

McKenna, T., Davis, J. and Zornetzer, S.F., editors, (1992) *Single Neuron Computation*, Academic Press, San Diego.

Mettala, E.G. (1989) *Automatic Generation of Control Software in Computer Integrated Manufacturing*, PhD Dissertation, The Pennsylvania State University, University Park.

Molloy, M.K. (1982) Performance Analysis Using Stochastic Petri Nets. *IEEE Transactions on Computers*, C-**31**(9), pp. 913-917.

Moore, K.E. and S.M. Gupta (1996) Petri Net Models of Flexible and Automated Manufacturing Systems: A Survey. *International Journal of Production Research*, **34**(11), pp. 3001-3035

Murata, T. (1989) Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, **77**, April, pp. 541-580.

Naylor, A. W. and Volz, R. A. (1987) Design of Integrated Manufacturing System Control Software. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-**17** (6), Nov/Dec, pp. 881-897.

Osakada, K. and Yang, G.B. (1991a) Neural Networks for Process Planning of Cold Forging. *Annals of the CIRP*, **40**(1), pp. 243-246.

Osakada, K. and Yang, G.B. (1991b) Application of Neural Networks to an Expert System for Cold Forging. *International Journal Machine Tools and Manufacture*, **31**(4), pp. 577-587.

Osakada, K., Yang, G.B., Nakamura, T. and Mori, K. (1990) Expert System for Cold Forging Process Based on FEM Simulation. *Annals of the CIRP*, **39**(1), pp. 249-252.

Panwalkar, S. S. and Iskander, W., (1977) A Survey of Scheduling Rules, *Operations Research*, **25**(1), pp. 45-61.

Peterson, J. L. (1981) *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs.

Petri C.A. (1962) Kommunikation mit Automaten, Ph.D. Dissertation, Bonn: Institute fur Instrumentelle Mathematik, Schriften des IIM Nr. 3. English Translation, "Communication with Automata, New York: Griffis Air Force Base, Tech Report RADC-TR-65-377, Vol. 1, Suppl. 1, 1966.

Phillips, D.T., and Garcia-Diaz A. (1981) *Fundamentals of Network Analysis*, Prentice Hall, Inc., Englewood Cliffs.

Ramchandani, C. (1974) Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, PhD dissertation, Massachusetts Institute of Technology, Cambridge.

Rogers, J. (1997) *Object-Oriented Neural Networks in C++*, Academic Press, Inc., San Diego.

Rosenfeld, S.A. (1992) *Competitive Manufacturing: New Strategies for Regional Development*, Center for Urban Policy Research, New Brunswick, NJ.

Senehi, M.K., Barkmeyer, E., Luce, M., Ray, S., Wallace, E., and Wallace, S. (1991) Manufacturing Systems Integration Initial Architecture Document, National Institute of Standards and Technology, NIST Interagency Report NISTIR 4682, Gaithersburg, MD.

Shinich, N., and Taketoshi, Y. (1992) Dynamic Scheduling System Utilising Machine Learning as a Knowledge Acquisition Tool. *International Journal of Production Research*, **30**, pp. 411-431.

Shukla, C.S., and Chen, F.F. (1996) The State of the Art in Intelligent Real-Time FMS Control: a Comprehensive Survey. *Journal of Intelligent Manufacturing*, **7**, pp. 441-455.

Simpson, J. A., Hocken, R. J., and Albus, J. S. (1982) The Automated Manufacturing Research Facility of the National Bureau of Standards, *Journal of Manufacturing Systems*, **1**(1), pp. 17-31.

Smith, J. S. (1992) A Formal Design and Development Methodology for Shop Floor Control in Computer Integrated Manufacturing, Ph.D. Dissertation, The Pennsylvania State University, University Park.

Smith, J. S., Cohen, P.H., Davis, J. W., and Irani, S.A. (1992) Process Plan Generation for Sheet Metal Parts Using an Integrated Feature-Based Expert System Approach. *International Journal of Production Research*, **30**(5), pp. 1175-1190.

Smith, J. S., Hoberecht, W. C. and Joshi, S.B. (1996) A Shop Floor Control Architecture for Computer Integrated Manufacturing. *IIE Transactions*, **28**(10), pp. 783-794.

Smith, J.S. and Joshi, S.B. (1995) A Shop Floor Controller Class for Computer Integrated Manufacturing. *International Journal of Computer Integrated Manufacturing*, **8**(5), September-October, pp. 327-339.

Smith, J.S. and Peters, B.A. (1998) Simulation as a Decision-Making Tool for Real-Time Control of Flexible Manufacturing Systems, in the *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98)*, Leuven, Belgium, May 16-20, 1998, IEEE Computer Society, pp. 586-590.

Smith, J.S., Wysk, R. A., Sturrock, D., Ramaswamy, S., Smith, G., and Joshi, S. B. (1994) Discrete Event Simulation for Shop Floor Control, in *Proceedings of the 1994 Winter Simulation Conference*, December 1994, Lake Buena Vista, FL, pp. 962-969.

Storer, R. H., Wu, S. D., and Vaccari, R. (1992) New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management Science*, **38**(10), October, pp. 1495-1509.

Storer, R. H., Wu, S. D., and Vaccari, R. (1995) Problem and Heuristic Space Search Strategies for Job Shop Scheduling. *ORSA Journal on Computing*, **7**(4), Fall, pp. 453-467.

Venugopal, V. and Narendran, T.T. (1992) Neural Network Model for Design Retrieval in Manufacturing Systems. *Computers in Industry*, **20**, pp.11-23.

Viswanadham, N., Narahari, Y. and Johnson, T.L. (1990) Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models. *IEEE Transactions on Robotics and Automation*, **6**(6), December, pp. 713-723.

Wall, M.B. (1996) A Genetic Algorithm for Resource-Constrained Scheduling, Ph.D. Dissertation, Massachusetts Institute of Technology, Cambridge.

Wysk, R. A., Peters, B. A., and Smith, J. S. (1995) A Formal Process Planning Schema for Shop Floor Control. *Engineering Design and Automation Journal*, **1**(1), Spring , pp. 3-19.

Wysk, R. A. and Smith, J. S. (1995) A Formal Functional Characterization of Shop Floor Control. *Computers in Industrial Engineering*, **28**(3), pp. 631-644.

Wysk, R. A., Yang, N.S., and Joshi, S. (1991) Detection of Deadlocks in Flexible Manufacturing Cells. *IEEE Transactions on Robotics and Automation*, **7**(6), December, pp. 853-859.

Wysk, R. A., Yang, N.-S., and Joshi, S. (1994) Resolution of Deadlocks in Flexible Manufacturing Systems: Avoidance and Recovery Approaches. *Journal of Manufacturing Systems*, **13**(2), pp. 128-138.

Zhang, B.-T., Ohm, P., and Muhlenbein, H. (1997) Evolutionary Induction of Sparse Neural Trees. *Evolutionary Computation*, **5**(2), pp. 213-236.

APPENDICES

APPENDIX A

THE PARABLE OF THE BLIND MEN AND THE ELEPHANT

American poet John Godfrey Saxe (1816-1887) based the following poem on a fable, which was told in India many years ago.

It was six men of Indostan
To learning much inclined,
Who went to see the Elephant
(Though all of them were blind),
That each by observation
Might satisfy his mind

The First approached the Elephant,
And happening to fall
Against his broad and sturdy side,
At once began to bawl:
"God bless me! but the Elephant
Is very like a wall!"

The Second, feeling of the tusk,
Cried, "Ho! what have we here
So very round and smooth and sharp?
To me 'tis mighty clear
This wonder of an Elephant
Is very like a spear!"

The Third approached the animal,
And happening to take
The squirming trunk within his hands,
Thus boldly up and spake:
"I see," quoth he, "the Elephant
Is very like a snake!"

The Fourth reached out an eager hand,
And felt about the knee.
"What most this wondrous beast is like
Is mighty plain," quoth he;
" 'Tis clear enough the Elephant
Is very like a tree!"

The Fifth, who chanced to touch the ear,
Said: "E'en the blindest man
Can tell what this resembles most;
Deny the fact who can
This marvel of an Elephant
Is very like a fan!"

The Sixth no sooner had begun
About the beast to grope,
Than, seizing on the swinging tail
That fell within his scope,
"I see," quoth he, "the Elephant
Is very like a rope!"

And so these men of Indostan
Disputed loud and long,
Each in his own opinion
Exceeding stiff and strong,
Though each was partly in the right,
And all were in the wrong!

Moral:
So oft in theologic wars,
The disputants, I ween,
Rail on in utter ignorance
Of what each other mean,
And prate about an Elephant
Not one of them has seen!

APPENDIX B

USER INPUT DATABASE TABLE FIELDS

Table 61 Equipment

Field Name	Data Type	Usage
EquipmentNumber	Integer	Identifies the piece of equipment
EquipmentDescription	Text	Easy human identifier
EquipmentType	Text	Classification: MP, MT, MH, BF, AS
ControllerName	Text	Where command messages should be sent

Table 62 FixedpLocations

Field Name	Data Type	Usage
LocationNumber	Integer	Identifies the location
LocationDescription	Text	Easy human identifier
EquipmentNumber	Integer	Identifies the equipment associated with the location

Table 63 IncompatibleTransporterMovements

Field Name	Data Type	Usage
PrimaryArc	Integer	The key field
IncompatibleArc	Integer	Arcs incompatible with the primary arc

Table 64 MobilepLocations

Field Name	Data Type	Usage
LocationNumber	Integer	Identifies the location
Location Description	Text	Easy human identifier
TransporterType	Integer	The transporter type associated with the location

Table 65 PartCarrierTypes

Field Name	Data Type	Usage
PartCarrierTypeNumber	Integer	Identifies the part carrier type
PartCarrierDescription	Text	Easy human identifier
TransporterType	Integer	Transporter Type the carrier can be used with

Table 66 PartID

Field Name	Data Type	Usage
PartNumber	Integer	Part Identification Number
PartName	Text	Short human recognizable name
PartDescription	Text	Longer human recognizable description

Table 67 PPArcs

Field Name	Data Type	Usage
ArcNumber	Integer	Identifies the arc
PartNumber	Integer	Identifies the part this arc applies to
StartingNode	Integer	Identifies the tail of the arc
EndingNode	Integer	Identifies the head of the arc

Table 68 PPNodes

Field Name	Data Type	Usage
PartNumber	Integer	Identifies the part type
NodeNumber	Integer	Identifies the Node of a given process plan
Equipmentnumber	Integer	Identifies what piece of equipment this node uses
Instructions	Text	Identifies the instruction file to be processed
EstimatedTime	Integer	Time required for the process in seconds

Table 69 ProcessingWorkstations

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identifies the workstation
Description	Text	Easy human identifier
ControllerName	Text	Where command messages should be sent

Table 70 ProcessingWSEquipAssn

Field Name	Data Type	Usage
Workstation Number	Integer	Identify the workstation
Equipment Number	Integer	Identify the equipment

Table 71 ProcessingWSLPAssn

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identify the workstation
TlocationNumber	Integer	Identify the Tlocation (Load Point)

Table 72 ProcessingWSMGArcs

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identifies the workstation the arc belongs to
ArcNumber	Integer	Identifies the Arc in the workstation
EquipmentNumber	Integer	Identifies the MH equipment that makes the move
EstimatedTime	Integer	How long the move will take in seconds
TypeofArc	Integer	What the arc is doing: 3=Unload, 2=Xfer, or 1=Load
PartOnly	Boolean	Boolean: True, only the part moves; False, means the part carrier moves with the part
LocationData1	Integer	Data entered in From To format content varies depending on the type of arc
LocationData2	Integer	May contain a fixed plocation, tlocation or mobile plocation
LocationData3	Integer	

Table 73 ProcessingWSUPAssn

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identify the workstation
TlocationNumber	Integer	Identify the Tlocation (Unload Point)

Table 74 StorageWorkstations

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identifies the workstation
Description	Text	Easy human identifier
ControllerName	Text	Where command messages should be sent

Table 75 StorageWSEquipAssn

Field Name	Data Type	Usage
Workstation Number	Integer	Identify the workstation
Equipment Number	Integer	Identify the equipment

Table 76 StorageWSLPAssn

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identify the workstation
TlocationNumber	Integer	Identify the Tlocation (Load Point)

Table 77 StorageWSMGArcs

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identifies the workstation the arc belongs to
ArcNumber	Integer	Identifies the Arc in the workstation
EquipmentNumber	Integer	Identifies the MH equipment that makes the move
EstimatedTime	Integer	How long the move will take in seconds
TypeofArc	Integer	What the arc is doing: 3=Unload, 2=Xfer, or 1=Load
PartOnly	Boolean	Boolean: True, only the part moves; False, means the part carrier moves with the part
LocationData1	Integer	Data entered in From To format content varies depending on the type of arc
LocationData2	Integer	May contain a fixed plocation, tlocation or mobile plocation
LocationData3	Integer	

Table 78 StorageWSUPAssn

Field Name	Data Type	Usage
WorkstationNumber	Integer	Identify the workstation
TlocationNumber	Integer	Identify the Tlocation (Unload Point)

Table 79 TLocations

Field Name	Data Type	Usage
LocationNumber	Integer	Identifies the location
LocationDescription	Text	Easy human identifier
EquipmentNumber	Integer	Identifies the equipment associated with the location
LoadPoint	Boolean	True/False is this a load point
UnLoadPoint	Boolean	True/False is this an unload point

Table 80 TMGArcs

Field Name	Data Type	Usage
ArcNumber	Integer	Identifies the arc
EquipmentNumber	Integer	Identifies the equipment that moves the transporter
EstimatedTime	Integer	Time for the move to complete (in seconds)
Startingtlocation	Integer	Identifies the tail of the arc
Endingtlocation	Integer	Identifies the head of the arc

Table 81 Transporters

Field Name	Data Type	Usage
TransporterNumber	Integer	Identifies the transporter
Type	Integer	Identifies the type of transporter
HomeLocation	Integer	the Tlocation number where the transporter starts in Empty and Idle conditions

Table 82 TransporterTypes

Field Name	Data Type	Usage
TransporterTypeNumber	Integer	Identifies the transporter type
PlocationCount	Integer	The number of plocations associated with this type
TransporterDescription	Text	Easy human identifier

Table 83 Parts

Field Name	Data Type	Usage
PartType	Long Integer	The part type identifer
ProcessNode	Long Integer	The part process plan node associated with this part
ProcessComplete	Boolean	Has the part completed processing at this node
PartCarrierType	Long Integer	The type of part carrier the part is attached to, zero indicates no carrier
StorageLocation	Long Integer	The FPL for the storage system

APPENDIX C

PRIMARY OUTPUT DATABASE TABLES AND FIELDS

The primary output database holds the Petri net and neural net information.

Table 84 List of Primary Output Data Tables

Table Name	Usage
BufferEmptyIndicatorIndex	Index of neural net node that indicate a buffer is empty
ControlData	Holds identification information
CurrentTokens	The tokens in the system
EmptyTokens	The tokens in the system if it is empty and idle
FPLIndex	Index to match the fixed part locations to the Petri net nodes representing availability and occupation
MHIndex	Index to match the material handlers to the Petri net node representing availability
NeuralNetLinks	The neural net links with properties
NeuralNetNodes	The neural net nodes with properties
OrderVector	The link between part numbers and order vector position
OVValues	The number of parts ordered
PartIndex	Links the part process plan nodes to Petri net processing nodes and fixed part locations
PNArc	The Petri net arcs
PNEvents	The Petri net events
PNMsg	The Petri net output message formats
PNNode	The Petri net nodes
SMColumnInfo	Status matrix column information
SMRowInfo	Status matrix row information
SMValues	Status matrix values (not used, originally meant to store the status matrix values)
TlocationIndex	The link between transporter locations and the Petri net nodes representing availability and occupation
TMGArcIndex	Transporter movement graph arc index matches arcs to Petri net nodes representing movement in progress and the decision input node that authorizes the movement
TokenCapacity	The capacity of the tokens
WSNeedsTransCapIndex	Index of nodes that indicate a workstation needs transportation capacity

Table 85 BufferEmptyIndicatorIndex Fields

Field Name	Data Type	Usage
BufferFPL	Integer	The fixed part location allocated to the buffer
BufferEmptyNNNode	Integer	The identifier of the neural net node that indicates the buffer is empty

Table 86 ControlData Fields

Field Name	Data Type	Usage
MyName	Text	The name the Petri net portion of the controller uses
MyBoss	Text	The controller authorized to send decision inputs
MySubordinate	Text	The name the neural net sends commands to
RouterName	Text	The name of the router
RouterHost	Text	The host the router is running on
RouterIP	Text	The IP address of the host the router is running on
RouterPort	Integer	The port the router is listening on

Table 87 CurrentTokens and EmptyTokens Fields

Field Name	Data Type	Usage
PNNode	Integer	The location of the token
TokenType	Integer	The token type
TransporterType	Integer	The transporter type only valid for type 1 tokens
CarrierType	Integer	The part carrier type only valid for type 3 tokens
PartType	Integer	The part type only valid for type 4 tokens
ProcessComplete	Integer	The current part node process is complete
ProcessNode	Integer	The current process plan node only valid for type 4 tokens
MobilePL	Integer	The mobile part location occupied
LastEventTime	Integer	The time the last event involving this token occurred
MTTime	Integer	Cumulative time spent in transport
MPTime	Integer	The cumulative time spent processing
MHTime	Integer	The cumulative time spent in material handling
BFTime	Integer	The cumulative time spent in buffers
ASTime	Integer	The cumulative time spent in storage excludes raw material and finished products
MPdelayTime	Integer	The cumulative time spent occupying a material processor not involved in processing
PartStartTime	Integer	The time the start command was issued
OrderArrivalTime	Integer	The time the order for the part arrived
SimpleFlowStartTime	Integer	The time the part was started used by type two tokens
SimpleFlowEndTime	Integer	The time the part becomes finished product used by type 2 tokens

Table 88 FPLIndex Fields

Field Name	Data Type	Usage
FPL	Integer	The fixed part location identifier
PNAvailable	Integer	The Petri net node indicating the fpl is available
PNHasPart	Integer	The Petri net node indicating a part is in the fpl
PNProcessing	Integer	The Petri net node indicating a part is processing at the fpl

Table 89 MHIndex Fields

Field Name	Data Type	Usage
MH	Integer	Material Handler Identifier
PNNode	Integer	Petri net node indicating the handler is available

Table 90 NeuralNetLinks Fields

Field Name	Data Type	Usage
IDNumber	Integer	Identification number for the link
InputNodeID	Integer	The node at the tail of the arc that provides an input value
OutPutNodeID	Integer	The node at the head of the arc that receives the output of the link
Weight	Double	The link multiplier value
LinkType	Integer	Indicates the function of the link

Table 91 NeuralNetNodes Fields

Field Name	Data Type	Usage
IDNumber	Integer	Node Identifier
Layer	Integer	The neural net layer the node lies in
SMRow	Integer	The status matrix row associated with the node only valid for layer 0 nodes
SMColumn	Integer	The status matrix column associated with the node only valid for layer 0 nodes
Threshold	Double	The minimum input value the node must have to generate a positive output
PNNodeAuthorized	Integer	The Decision Input Petri net node that precedes the activity associated with this node
PNProcessingNode	Integer	The standard place Petri net node that represents the ongoing activity associated with this node
IsOrderVector	Boolean	Is this node part of the order vector only valid for layer 0 nodes
OrderVectorPosition	Integer	The order vector index position associated with this node only valid for layer 0 nodes
Message	Text	The message that will be sent to the Petri net if this node is activated, only valid for output layer nodes
Usage	Integer	Indicates what the neural net node does, used for optimization purposes

Table 92 OrderVector Fields

Field Name	Data Type	Usage
OrderVectorIndex	Integer	The index into the order vector
PartIDNumber	Integer	The part represented by this element of the order vector

Table 93 OVValues Fields

Field Name	Data Type	Usage
VectorPosition	Integer	The index into the order vector
Value	Integer	The number of parts to be created

Table 94 PartIndex Fields

Field Name	Data Type	Usage
PartType	Integer	Part type identifier
ProcessNode	Integer	Process process plan node identifier
ProcessComplete	Boolean	Indicates whether the part is complete
PetriNetNode	Integer	The Petri net node associated with the process plan node and completion status
ProcessingFPL	Integer	The fixed part location where processing takes place

Table 95 PNArc Fields

Field Name	Data Type	Usage
Number	Integer	Petri net arc identifier
Tail	Integer	The tail / origin of the arc
Head	Integer	The head / destination of the arc
Color	Integer	The arc type

Table 96 PNEvents Fields

Field Name	Data Type	Usage
TransitionNumber	Integer	The identifier of the transition that will fire when the event occurs
Message	Text	The format for the message that will be received
Controller	Text	The source of the message that will be received
Conversions	Integer	The number of parameters that must be retrieved from the incoming message

Table 97 PNMsg Fields

Field Name	Data Type	Usage
Number	Integer	The Petri net output node identifier
Controller	Text	The destination of the message
Msg	Text	The format of the message that must be sent

Table 98 PNNode Fields

Field Name	Data Type	Usage
Number	Integer	Identifier of the Petri net node
Type	Integer	The Petri net node type
SMCol	Boolean	Is this node associated with a status matrix column
SMRow	Boolean	Is this node associated with a status matrix row
Deadlock	Boolean	Is this node used for deadlock detection
TimeCategory	Integer	Indicates the where the time spent in this node should be assigned

Table 99 SMColumnInfo Fields

Field Name	Data Type	Usage
StatusMatrixColumn	Integer	The status matrix column index
TokenType	Integer	The type (or color) of the token
TransporterType	Integer	The type of transporter if token type = 1
CarrierType	Integer	The type of part carrier if token type = 3
PartType	Integer	The part type if token type = 4
ProcessComplete	Boolean	Has the part completed the processing at the node
ProcessNode	Integer	The part process plan node if token type = 4

Table 100 SMRowInfo Fields

Field Name	Data Type	Usage
StatusMatrixRow	Integer	The status matrix row index
PNNode	Integer	The Petri net node associated with the status matrix row
DeadlockFlag	Boolean	Is this row used for deadlock detection

Table 101 SMValues Fields

Field Name	Data Type	Usage
SMRow	Integer	The status matrix row index
SMCol	Integer	The status matrix column index
Value	Integer	The value of the status matrix element

Table 102 TlocationIndex Fields

Field Name	Data Type	Usage
Tlocation	Integer	The transporter location identifier
TlocAvailable	Integer	The Petri net node that indicates the transporter location is available
TLocHasT	Integer	The Petri net node that indicates the transporter location is occupied by a transporter
TLocNNT3CapAvail	Integer	The identifier of the neural net node that indicates the transporter location is occupied by a transporter with type 3 capacity available
TLocNNT4CapAvail	Integer	The identifier of the neural net node that indicates the transporter location is occupied by a transporter with type 4 capacity available
TLocNNOccupied	Integer	The identifier of the neural net node that indicates the transporter location is occupied by a transporter
TLocHasPart	Integer	The identifier of the neural net node that indicates the transporter location is occupied by a transporter that contains a part

Table 103 TMGArcIndex Fields

Field Name	Data Type	Usage
TMGArc	Integer	The transporter movement graph arc
ProcessingNode	Integer	The Petri net node that indicates a transporter is moving along the arc
AuthorizedNode	Integer	The Petri net decision input node that authorizes movement along the arc

Table 104 TokenCapacity Fields

Field Name	Data Type	Usage
TokenType	Integer	The token type
ItemType	Integer	The transporter or part carrier type
Capacity	Integer	The number of items that can be placed on the transporter or part carrier

Table 105 WSNeedsTransCapIndex Fields

Field Name	Data Type	Usage
WSNumber	Integer	The workstation identifier
WSIsStorage	Boolean	True if the workstation is a storage workstation
Type3NodeNumber	Integer	Identifier of the neural net node that indicates the workstation needs type 3 transport capacity
Type4NodeNumber	Integer	Identifier of the neural net node that indicates the workstation needs type 4 transport capacity
BlockedEmptyNodeNumber	Integer	Identifier of the neural net node that indicates a part trying to reach the workstation is blocked by empty transporters

APPENDIX D

EXEMPLAR DATABASE TABLES AND FIELDS

The Exemplar database holds the training data.

Table 106 List of Exemplar Data Tables

Table Name	Usage
ChoicePointsChoices	Identifies the possible choices at a choice point
ChoicePointsID	Identifies the choice points
DeadlockBeginEndLocations	Outdated. Stored data used in the first deadlock recovery method
DeadlockIdentification	Outdated. Same as Identification, but used for data generated during deadlock recovery
DeadlockInputValues	Outdated. Same as InputValues, but used for data generated during deadlock recovery
DeadlockOutputValues	Outdated. Same as OutputValues, but used for data generated during deadlock recovery
DeadlockPathSteps	Outdated. Same as EquipmentPaths, but used for data generated during deadlock recovery
EquipmentPaths	Paths through the workcell, there are one or more equipment based paths for each process plan path
EquipPathPerformance	The time required to complete the equipment path
GenomeChoicePointValues	Holds the values assigned to a choice point by a genome
GenomeID	Holds the genome identification and performance data
GenomeInhibitChoicePointValues	Holds the values assigned to an inhibit choice point by a genome
Identification	Identification data for each exemplar data point
InhibitChoicePointsChoices	Identifies the possible choices at an inhibit choice point
InhibitChoicePointsID	Identifies the inhibit choice points
InputValues	The input portion of the exemplar data
L3Incompatibility	Lists incompatibility between preliminary output nodes
L3toL4map	Matches the preliminary output node to the corresponding final output node
MovementPaths	Equipment paths for empty transporters
MovementPathPerformance	The time required to complete the movement path
NeuralNetResults	Outdated. Held data used in the original neural net training scheme
OutputValues	The output portion of the exemplar
ProcessPlanPath	Paths through the process plan from the raw material to the finished product node
TrainingParameters	Outdated. Held data used to define the neural net training procedure

Table 107 ChoicePointsChoices Fields

Field Name	Data Type	Usage
IDNum	Integer	The identification of the choice point
NNNode	Integer	The neural net node to set to the minimum threshold
ChoiceIDNum	Integer	The identification of the choice for this choice point
OutputNNNode	Integer	The preliminary output node activated by this choice

Table 108 ChoicePointsID Fields

Field Name	Data Type	Usage
IDNum	Integer	The identification number of the choice point
NumberOfChoices	Integer	The number of possible choices
MinimumThreshold	Double	The minimum threshold to assign to the nodes associated with the choice point
Description	Text	A description of the choice point

Table 109 DeadlockBeginEndLocations Fields

Field Name	Data Type	Usage
DeadlockPathNumber	Integer	Path identification number
OriginLocation	Integer	Path start location
OriginIsFPL	Boolean	True if path starts at a fixed part location
DestinationLocation	Integer	Path end location
DestinationIsFPL	Boolean	True if path ends at a fixed part location
PreferredUnloadTLocation	Integer	Preferred unload point if the path starts at a fixed part location

Table 110 EquipmentPaths and DeadlockPathSteps Fields

Field Name	Data Type	Usage
PathNumber	Integer	Equipment path identifier
PathStep	Integer	Path step identifier
PartType	Integer	The type of part being processed
ProcessNode	Integer	The process plan node
LocationIdentifier	Integer	The location where the part is located
LocationIsFPL	Boolean	Is the location a fixed part location
Command	Text	The command that needs to be sent

Table 111 EquipPathPerformance Fields

Field Name	Data Type	Usage
EquipmentPath	Integer	Equipment path identifier
Operations	Integer	The number of operations in the path
Length	Integer	The length of time the path requires

Table 112 GenomeChoicePointValues and GenomeInhibitChoicePointValues Fields

Field Name	Data Type	Usage
IDNum	Integer	The genome identification number
Choicepoint	Integer	The choice point identification number
TheChoice	Integer	The choice to be selected for this choice point

Table 113 GenomeID Fields

Field Name	Data Type	Usage
IDNum	Integer	The genome identification number
PerformanceValue	Integer	The objective value function for this genome
NumofChoicePts	Integer	The number of choice points used with this genome
NumofInhibitChoicePts	Integer	The number of inhibit choice points used with this genome

Table 114 Identification and DeadlockIdentification Fields

Field Name	Data Type	Usage
Number	Integer	Identification number for the exemplar
InputSize	Integer	The number of inputs used
OutputSize	Integer	The number of outputs generated
EpathNumber	Integer	The equipment path associated with the exemplar

Table 115 InhibitChoicePointsChoices Fields

Field Name	Data Type	Usage
IDNum	Integer	The choice point identification number
ArcNumber	Integer	The arc that should have its weight set to zero
ChoiceIDNum	Integer	The identification number for this choice

Table 116 InhibitChoicePointsID Fields

Field Name	Data Type	Usage
IDNum	Integer	Identification number for the inhibit choice point
NumberOfChoices	Integer	The number of choices possible
Description	Text	A description of the inhibit choice point generally includes the messages that were conflicting

Table 117 InputValues and DeadlockInputValues Fields

Field Name	Data Type	Usage
Number	Integer	The exemplar this input value belongs to
NNNode	Integer	The neural net node that is used as the input
SMRow	Integer	The status matrix row associated with the neural net node
SMCol	Integer	The status matrix column associated with the neural net node
Value	Double	The value of the neural net node input

Table 118 L3Incompatibility Fields

Field Name	Data Type	Usage
PrimaryL3Node	Integer	The node being considered
IncompatibleL3Node	Integer	A node that conflicts because of common resource usage

Table 119 L3toL4mapFields

Field Name	Data Type	Usage
L3Node	Integer	Preliminary output node
L4Node	Integer	Matching final output node

Table 120 MovementPaths Fields

Field Name	Data Type	Usage
PathNumber	Integer	The movement path identifier
PathStep	Integer	The step in the movement path
LocationIdentifier	Integer	The location identifier (will always be a Tlocation)
Command	Text	The command that needs to be sent

Table 121 MovementPathPerformance Fields

Field Name	Data Type	Usage
MovementPath	Integer	The movement path identifier
Operations	Integer	The number of operations in the path
Length	Integer	The length of time the path requires

Table 122 NeuralNetResults Fields

Field Name	Data Type	Usage
PatternNumber	Integer	The pattern (exemplar) being trained
OutputNumber	Integer	The output node number
Cycle	Integer	The training cycle
Repetition	Integer	The repetition in the training cycle
Value	Double	The value the output node was outputting

Table 123 OutputValues and DeadlockOutputValues Fields

Field Name	Data Type	Usage
Number	Integer	The exemplar this output is attached to
NNNode	Integer	The neural net output layer node
Value	Double	The desired value of the output node

Table 124 ProcessPlanPath Fields

Field Name	Data Type	Usage
PartNumber	Integer	The part number the path applies to
PathNumber	Integer	The path number (not part specific)
Path	Text	The path as a comma delimited node list

Table 125 TrainingParameters Fields

Field Name	Data Type	Usage
LearningRate	Double	Back propagation training parameter
Momentum	Double	Back propagation training parameter
MaxCycles	Integer	Maximum number of cycles to train
PatternToRepeat	Integer	The identification number of a single patter
RepsPerCycle	Integer	Number of replications per training cycle
RepeatSinglePattern	Boolean	True use only one pattern from the set
PrelimaryOutputLayer	Integer	The neural net preliminary output layer
SaveResults	Integer	How often results should be saved to the database

APPENDIX E

PROCESSING WORKSTATION PETRI NET GROWTH

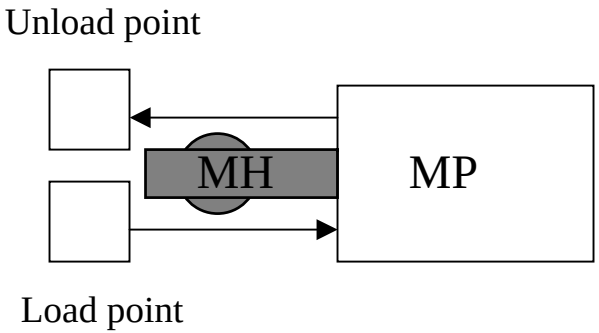

Figure 23 Simple Processing Workstation

Figure 24 Step 3 Add a Node for the MH

Figure 25 Step 4 Add Nodes for FPL

Figure 26 Step 5 Add Processing Activity

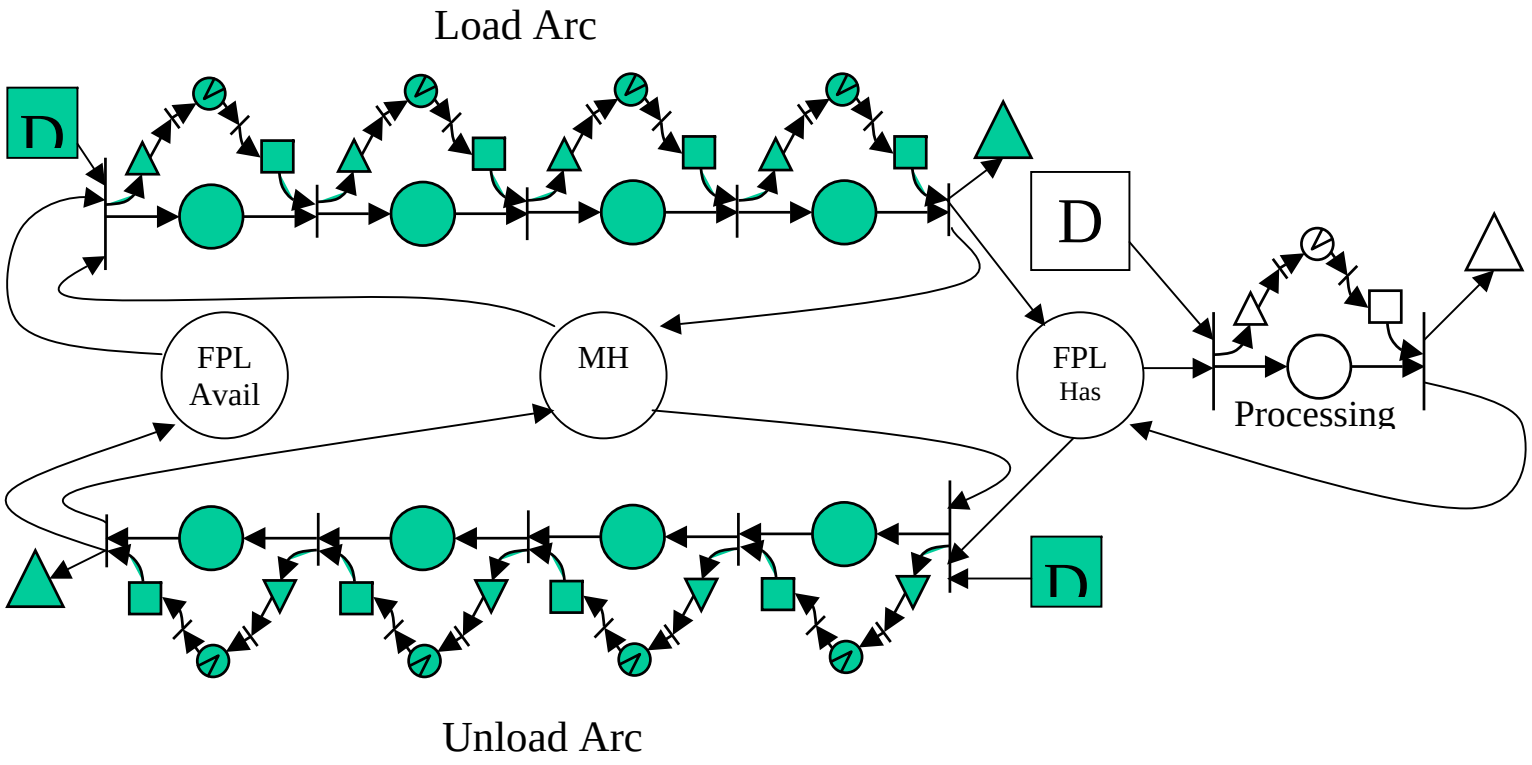

Figure 27 Step 6 Add Activities for WSMG Arcs

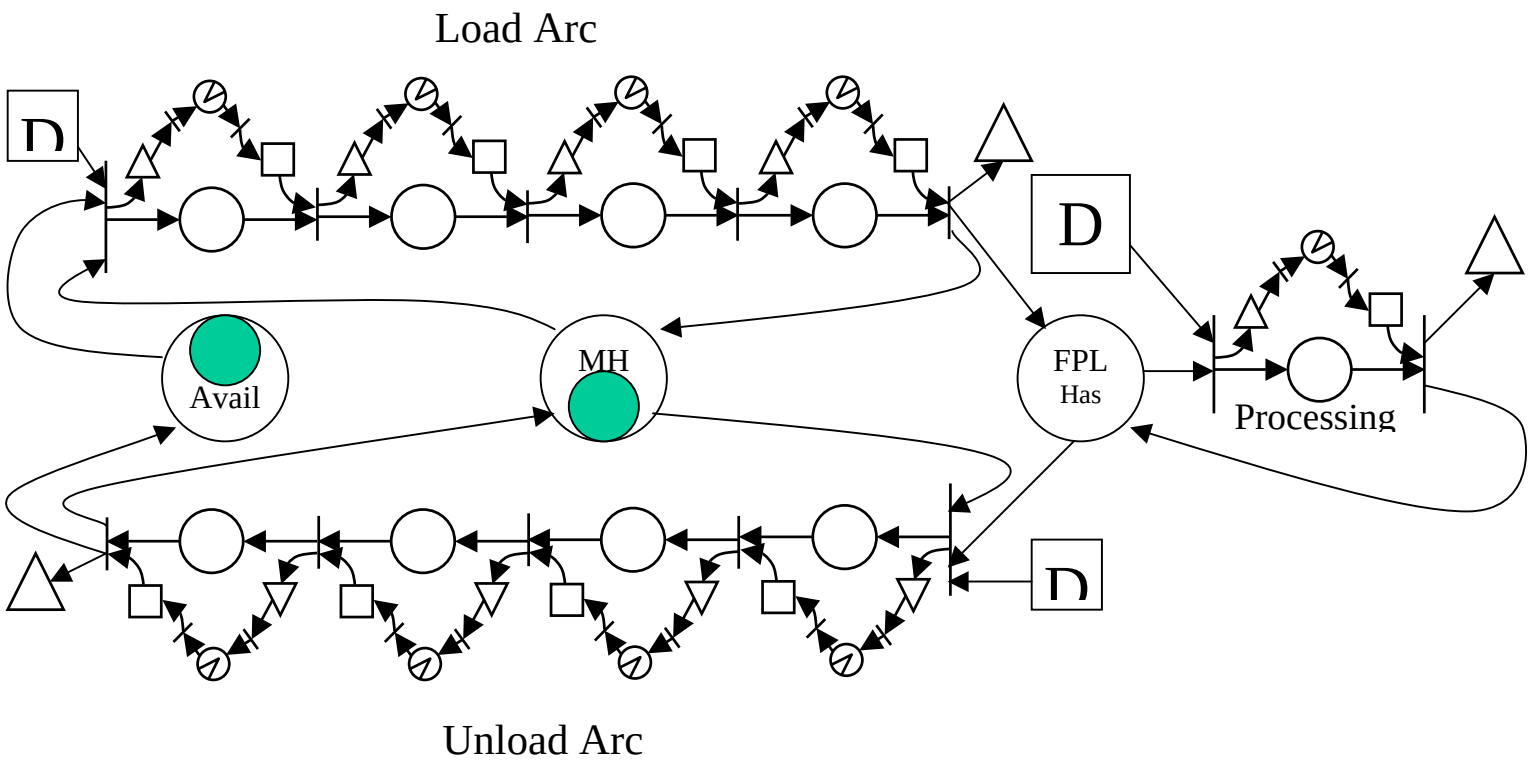

Figure 28 Step 7 Add Tokens

APPENDIX F

MANUAL LOGIC DEVELOPMENT TEST CASE ONE

Because test case one involved a single machine it is known that scheduling parts using the shortest processing time first heuristic will generate a schedule with the minimum mean flowtime. The neural net structure required to implement a shortest processing time first logic was constructed starting with a neural net that had the decision input control logic rules already in it (i.e. the output of the cell controller building program prior to any exemplar based construction).

The process plans were analyzed to find the number of paths possible for each part type and the path with the shortest processing time (see Table 126). Based on these results the parts need to be given priority in the following order: 2, 1, 4, 3. The following movement priority was used (from highest to lowest): load the processing workstation, unload the processing workstation, unload the storage workstation, load the storage workstation. The movement priority was used in developing the logic for the controller but the availability of the parts as they flowed through the system meant that there was never a time when the movement priority had to be enforced.

When referring to parts in the text below a 2-tuple of part type and process plan node will be used: (type, node).

Table 126 Process Plan Path Analysis Results

Part Type	Number of Paths	Minimum processing time	Minimum time path
1	2	600 seconds	1, 3, 2
2	1	400 seconds	1, 3, 2
3	2	800 seconds	1, 3, 2
4	2	675 seconds	1, 4, 5, 2

Start Logic

An initial part start logic was developed. A neural net node was added for each part type to indicate whether the part had been ordered or not (see Table 127). These nodes were supposed to turn the start

output on. The decision input rules built by the controller building program would prevent the output from being on if the number of parts previously started was equal to or greater than the number ordered or the raw material was not available.

Table 127 Initial Start Logic Nodes

Node Number	Usage
714	Part type 2 has been ordered
715	Part type 1 has been ordered
716	Part type 4 has been ordered
717	Part type 3 has been ordered

These start logic nodes (see Table 127) were connected to the order vector input nodes and the output nodes associated with part starting events as shown in Table 128. This logic was flawed. A part could not be started if a higher priority part had been ordered, even if the higher priority part was already completed. The links between the intermediate nodes and final nodes were removed (the intermediate nodes and the links to them could also have been removed but were not).

Table 128 Initial Start Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
1 -- Part 2 Order Vector	1, 3	714	1, 3	688 Start P2, N3
		714	-1, 3	684 Start P1, N3
		714	-1, 3	696 Start P4, N4
		714	-1,3	690 Start P3, N3
0 -- Part 1 Order Vector	1, 3	715	1, 3	684 Start P1, N3
		715	-1, 3	696 Start P4, N4
		715	-1, 3	690 Start P3, N3
3 -- Part 4 Order Vector	1, 3	716	1, 3	696 Start P4, N4
		716	-1, 3	690 Start P3, N3
2 -- Part 3 Order Vector	1, 3	717	1, 3	690 Start P3, N3

The controller building program created nodes that were designed to be true (high) when it was appropriate to start a part. These nodes were then connected to the output nodes with an inhibit if low

arc. A new start logic was developed using these nodes (Table 129). The inhibit low link was converted to a fixed weight excitatory link so the output would be triggered when the node was high. Priority was then enforced with a series of inhibit high links. Nodes 708, 711, and 712 were logic for alternate process plan paths that were not used so their logic was not altered and is not shown in Table 129. This logic corresponds to the following rule: IF *the number of parts of type N that have been ordered is greater than the number of parts of type N that have been started* AND *there is no higher priority part type ready to be started* THEN *start the part of type N.*

Table 129 Revised Start Logic

Generated Node	Link weight, type	Output Node	Function
707	1, 3	684	Start Type 1 Node 3 Okay
	1, 1	690	Inhibit starting type 3
	1, 1	696	Inhibit starting type 4
709	1, 3	688	Start Type 2 Node 3 Okay
	1, 1	684	Inhibit starting type 1
	1, 1	690	Inhibit starting type 3
	1, 1	696	Inhibit starting type 4
710	1, 3	690	Start Type 3, Node 3 Okay
713	1, 3	696	Start Type 4, Node 4 Okay
	1, 1	690	Inhibit starting type 3

Processing Logic

The processing logic rule used was: IF *there is a part in the processing workstation ready to process* THEN *process it.* The processing workstation was fixed part location 2 and was represented by status matrix row 3. The only parts that would be ready for processing were: (1,3), (2,3), (3,3), (4,4), (4,5), corresponding to status matrix columns 4, 9, 12, 18 and 19. The neural net nodes corresponding to this status matrix row and these columns were connected to a new neural node (718) with an excitatory link. This new node was then connected to the neural net output node that started the material processor. Because the Petri net stores the information about the part that is located at the material processor and there can be only one the neural net does not need to provide this information and uses only one process start message per material processor.

Table 130 Processing Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
68	1, 3	718	1, 3	604
73	1, 3	718		
76	1, 3	718		
82	1, 3	718		
83	1, 3	718		

Processing Workstation Unload Logic

Because the Petri net stores the information about the part that is located at the material processor and there can be only one the neural net does not need to provide this information for unload commands and uses only one message per unload arc. The processing workstation unload logic rule used was: IF *there is a part in the processing workstation ready to unload* THEN *unload it*. No priority is required because the workstation has a capacity of one. The parts that would be ready to unload were: (1,2), (2,2), (3,2), (4,2) corresponding to status matrix columns 3, 8, 11, and 16. The neural net nodes corresponding to the workstation status matrix row (3) and these columns were connected to a new neural node (719) with an excitatory link. This new node was then connected to the neural net output node that started the unload.

Table 131 Processing Workstation Unload Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
67	1, 3	719	1, 3	626
72	1, 3	719		
75	1, 3	719		
80	1, 3	719		

Storage Workstation Unload Logic

The storage workstation unload logic rule used was: IF *there is a part in the storage workstation ready to unload* AND *it is the highest priority part* THEN *unload it*. This logic is in addition to the decision input place base logic that will not allow an unload to occur if the unload destination is occupied. This translated into the following set of rules.

1. IF *there is a (2,3)* THEN *unload it.*
2. IF *there is a (1,3)* AND *there is not a (2,3)* THEN *unload it.*
3. IF *there is a (4,5)* AND *there is not a (2,3) or (1,3)* THEN *unload it.*
4. IF *there is a (4,4)* AND *there is not a (2,3), (1,3) or (4,5)* THEN *unload it.*
5. IF *there is a (3,3)* AND *there is not a (2,3), (1,3), (4,5) or (4,4)* THEN *unload it.*

The controller was not designed to create any (4,5) parts, type 4 parts that had only one processing step completed. However, by including the (4,5) part in the unload priority any existing parts of this type would be completed.

A new node was created for each part type node combination to be unloaded (see Table 132). The storage workstation was fixed part location one and was represented by status matrix row 2. The status matrix columns of interest were 9, 4, 19, 18, and 12 corresponding to (2,3), (1,3), (4,5), (4,4) and (3,3) respectively. A fixed weight excitatory arc was connected from the input layer node representing each part type to the new node for that part type and a second fixed weight excitatory node connected the new node to the output node that started the unload operation for that part type. Priorities were then enforced by connecting the intermediate node of each part to the output nodes of the lower priority parts with inhibit when high links (see Table 133).

Table 132 Storage Workstation Unload Logic Nodes

Node number	Part type and node
720	(2,3)
721	(1,3)
722	(4,5)
723	(4,4)
724	(3,3)

Table 133 Storage Workstation Unload Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
53	1,3	720	1,3	666
		720	1,1	658
		720	1,1	682
		720	1,1	680
		720	1,1	670
48	1,3	721	1,3	658
		721	1,1	682
		721	1,1	680
		721	1,1	670
63	1,3	722	1,3	682
		722	1,1	680
		722	1,1	670
62	1,3	723	1,3	680
		723	1,1	670
56	1,3	724	1,3	670

Processing Workstation Load Logic

The processing workstation load logic rule used was: IF *there is a part at the load point ready to load* THEN *load it*. No priority is required because the transporters have a capacity of one. The parts that would be ready for to load were: (1,3), (2,3), (3,3), (4,4), (4,5), corresponding to status matrix columns 4, 9, 12, 18 and 19. The load point was transporter location two represented by row one of the status matrix. The corresponding neural net input layer nodes were: 28, 33, 36, 42, 43. The input layer node for each part was connected to a new intermediate node (one per part type) by a fixed weight excitatory link. The intermediate node was then connected to the output node that sent the appropriate load command.

Table 134 Processing Workstation Load Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
33	1,3	725	1,3	612
43	1,3	726	1,3	624
28	1,3	727	1,3	606
42	1,3	728	1,3	622
36	1,3	729	1,3	614

Storage Workstation Load Logic

The processing workstation load logic rule used was: IF *there is a part at the load point ready to load* THEN *load it*. No priority is required because the transporters have a capacity of one. The parts that would be ready for to load were: (1,2), (2,2), (3,2), (4,2), corresponding to status matrix columns 3, 8, 11, and 16. The load point was transporter location one represented by row zero of the status matrix. The corresponding neural net input layer nodes were: 7, 12, 15, 20. The input layer node for each part was connected to a new intermediate node (one per part type) by a fixed weight excitatory link. The intermediate node was then connected to the output node that sent the appropriate load command.

Table 135 Processing Workstation Load Logic

Input Node	Link weight, type	Intermediate node	Link weight, type	Final node
12	1,3	730	1,3	636
7	1,3	731	1,3	628
20	1,3	732	1,3	648
15	1,3	733	1,3	640

APPENDIX G

ALGORITHM FOR GENERATING NEURAL NET LOGIC FROM PETRI NET DECISION INPUT PLACES

- Select all of the Decision Input Places (Petri net nodes with type = 4)
- For each decision input place find the event that triggers the transition preceding the place.
- Call the appropriate function based on the type of message that triggers the transition

Move

4. Add a node to the first hidden layer (threshold = 0.9)
5. Determine the status matrix row that represents the location of the move origin
6. Determine the status matrix columns that represent transporters
7. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer nodes that correspond to the status matrix row and columns found to the hidden layer node that was added
8. Identify those status matrix rows that represent incompatibilities with the move, these include other moves with the same destination in progress, the destination location has a transporter, a transporter at the destination location is involved in a load or unload operation
9. Add an inhibit high link to the node added in step one from the input layer nodes that represent the rows found in step five and the columns found in step three
10. Add an inhibitory (weight = 1, type = 1) link from the node added in step one to the output layer nodes that send messages that trigger the transition

Load

1. Add a node to the first hidden layer (threshold = 1.8)
2. Determine the status matrix row that represents the transporter location of the load operation
3. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer nodes that correspond to the status matrix row and the columns representing transporters to the hidden layer node that was added in step 1
4. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer nodes that correspond to the status matrix row and the columns representing part carriers to the hidden layer node that was added in step 1
5. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer nodes that correspond to the status matrix row and the columns representing parts to the hidden layer node that was added in step 1
6. Determine the status matrix row that represents the fixed part location that is the destination of the load operation
7. Add an inhibitory (weight = 1, type = 1) link from the input layer nodes that correspond to the status matrix row and the columns representing parts to the hidden layer node that was added in step 1
8. Determine if the destination is a material processor, if it is, find the status matrix row that represents the processing activity and add an inhibitory (weight = 1, type = 1) link from the input layer nodes that correspond to the status matrix row and the columns representing parts to the hidden layer node that was added in step 1

9. Determine the status matrix rows of the other activities that use the same material handler as the load operation and add an inhibitory (weight = 1, type = 1) link from the input layer nodes that correspond to the status matrix row and the columns representing either parts or part carriers depending on the Petri net arc associated with the activity to the hidden layer node that was added in step 1
10. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the load operation from the hidden layer node that was added in step 1

Unload

1. Add a node to the first hidden layer (threshold = 0.9) to represent a part ready to unload
2. Add a node to the first hidden layer (threshold = 0.9) to represent a transporter ready to receive a part
3. Add a node to the second hidden layer (threshold =1.8) to combine the outputs of the nodes added in steps 1 and 2
4. Add a fixed weight excitatory (weight = 1, type = 3) link from the nodes added in steps 1 and 2 to the node added in step 3
5. Determine the status matrix rows of the other activities that use the same material handler as the load operation and add an inhibitory (weight = 1, type = 1) link from the input layer nodes that correspond to the status matrix row and the columns representing either parts or part carriers depending on the Petri net arc associated with the activity to the hidden layer node that was added in step 3
6. Determine the status matrix row that represents the transporter location where the unload operation will terminate
7. Determine the columns that represent transporters and add a fixed weight excitatory (type = 3) link to the node added in step 2, where the weight equals the transporter capacity, from the input neural nodes that correspond to the row found in step 6 and the columns found in this step
8. Determine the columns that represent part carriers and add a fixed weight excitatory (type = 3) link to the node added in step 2, where the weight equals the part carrier capacity minus one, from the input neural nodes that correspond to the row found in step 6 and the columns found in this step
9. Determine the columns that represent parts and add a fixed weight excitatory (type = 3) link to the node added in step 2, where the weight equals minus one, from the input neural nodes that correspond to the row found in step 6 and the columns found in this step
10. Determine the status matrix row that represents the fixed part location where the unload originates
11. Determine whether part carriers are involved in this unload operation. If part carriers are involved add a set of links from the input neural nodes that correspond to the row found in step 10 and the columns representing part carriers, if part carriers are not involved then use the columns representing parts.
12. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the unload operation from the hidden layer node that was added in step 3

Transfer

1. Add a node to the first hidden layer (threshold = 0.9)
2. Determine the status matrix row that represents the destination fixed part location.
3. Determine the status matrix columns that represent parts or part carriers
4. Add inhibit arcs (weight = 1, type = 1) from the input layer neural net nodes representing the status matrix row found in step 2 and the columns found in step 3 to the node added in step 1

5. Determine if the destination fixed part location is a material processor, if so find the status matrix row that represents a part being processed.
6. Add inhibit arcs (weight = 1, type = 1) from the input layer neural net nodes representing the status matrix row found in step 5 and the columns representing parts to the node added in step 1
7. Determine the status matrix rows of all other activities that involve the material handler
8. Determine the whether to use part or part carrier columns (based on Petri net arc type)
9. Add inhibit arcs (weight = 1, type = 1) from the input layer neural net nodes representing the status matrix row found in step 7 and the columns found in step 8 to the node added in step 1
10. Determine the status matrix row that represents the origin FPL
11. Determine the status matrix columns that represent parts or part carriers (based on Petri net arc type)
12. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer neural net nodes representing the status matrix row found in step 10 and the columns found in step 11 to the node added in step 1
13. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the transfer operation from the hidden layer node that was added in step 1

Transform

1. Add a node to the first hidden layer (threshold = 0.9)
2. Determine the status matrix column associated with the transform being processed
3. Determine the fixed part location associated with the part / node combination to be transformed and use it to find the status matrix row associated with the transform
4. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer neural net node representing the status matrix row found in step 4 and the columnsfound in step 2 to the node added in step 1
5. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the transform operation from the hidden layer node that was added in step 1

Start

1. Add a node to the first hidden layer (threshold = 0.9)
2. Determine the status matrix row associated with the fixed part location associated with the start message (there will be one for each storage location)
3. Determine the status matrix column that represents the raw material
4. Add an inhibit low (weight = 1, type = 2) link to the neural net node added in step 1 from the input neural node that represents the status matrix row found in step 2 and the status matrix column found in step 3
5. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer neural net node representing the order vector for this part type
6. Determine the status matrix columns for all stages of the part excluding raw material
7. Add a fixed weight excitatory (weight = -1, type = 3) link from the input layer neural net nodes representing all status matrix rows and the columns found in step 6.
8. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the start operation from the hidden layer node that was added in step 1

Process

1. Add a node to the first hidden layer (threshold = 0.9)
2. Determine the status matrix row associated with the fixed part location where the part will be processed

3. Add a fixed weight excitatory (weight = 1, type = 3) link from the input layer neural net nodes representing the status matrix row and the columns representing parts to the node added in step 1
4. Add an inhibit if low (weight = 1, type = 2) link to the neural net output nodes that trigger the start operation from the hidden layer node that was added in step 1

APPENDIX H

WORKCELL USER INPUT TRANSLATION ALGORITHM

Algorithm
Steps one through ten represent the creation of the equipment-based portion of the workcell controller. See Process Plan Conversion Algorithm for the creation of the rest of the workcell controller.

1. For each TLocation add two standard places (type = 2). The first one signals the location is available for a transporter to move into it. The second one signals when a transporter in the location can move. Update the TLocationIndex table.

2. Add the material handler available nodes. "Select Equipment Number from Equipment where Equipment Type = MH" For each MH add one standard place (type = 2) and update the MHIndex.

3. Add the fixed part locations. Need to add a standard place (type = 2) for all plocations that are associated with MP and BF equipment. Add a high capacity place for plocations (type = 7) associated with automated storage equipment. Create recordset rs1 using "Select * from Fixed Plocations" to get all fixed part locations. Then create recordset rs2 using "Select [Equipment Type] from Equipment where [Equipment Number] = rs1.[Equipment Number]" if rs2.[Equipment Type] = AS then add type 7 else add type 2. Update the FPLIndex.

4. Process transporter movement graph arcs. For each arc
   Add a transition (type = 1)
   Setup pre-conditions
      Add a Decision input place (type = 4)
      Add an arc from the decision input place to the transition
      Add an event transition, message = MOVE,starting tlocation, ending tlocation
      Add an arc from the event transition to the decision input place
      Add an arc from the ending tlocation available place to the transition
      Add an arc from the starting tlocation hasT place to the transition
   Add Activity(equip_to_Controller, "MOVE,start tloc, end tloc")
   Setup post-conditions
      Add arc from Activity complete transition to starting tlocation available place
      Add arc from Activity complete transition to ending tlocation hasT place

5. Setup MP processing loops.
   Add a transition (type = 1)
   Setup pre-conditions
      Add a Decision input place (type = 4)
      Add an arc from the decision input place to the transition
      Add an event transition, message = PROCESS, fpl
      Add an arc from the event transition to the decision input place
      Add an arc from the fpl has part place to the transition
   Add Activity (fpl_to_WS, PROCESS,fpl, Type= %i,Node= %i)
   Setup post-condtions
      Add an arc from the Activity complete transition to the fpl has part place

6. Process processing workstation movement graph arcs
   Load

Add a transition (type = 1)
Setup pre-conditions
Add a Decision input place (type = 4)
Add an arc from the decision input place to the transition
Add an event transition, message = LOAD,starting tlocation, ending fpl
Add an arc from the event transition to the decision input place
Add an arc from the ending fpl available place to the transition
Add an arc from the starting tlocation hasT place to the transition
Add an arc from the MH available place to the transition
Add Activity (fpl_to_WS, LOAD,tloc,%I, fpl, Type= %i,Node= %i)
Setup post conditions
Add an arc from the Activity complete transition to the fpl has part place
Add an arc from the Activity complete transition to the MH available place
Add an arc from the Activity complete transition to the starting tlocation hasT place

Unload
Add a transition (type = 1)
Setup pre-conditions
Add a Decision input place (type = 4)
Add an arc from the decision input place to the transition
Add an event transition, message = UNLOAD,starting fpl, ending tlocation
Add an arc from the event transition to the decision input place
Add an arc from the starting fpl has part place to the transition
Add an arc from the ending tlocation hasT place to the transition
Add an arc from the MH available place to the transition
Add Activity (fpl_to_WS, UNLOAD, fpl, tloc,%I, Type= %i,Node= %i)
Setup post conditions
Add an arc from the Activity complete transition to the fpl available place
Add an arc from the Activity complete transition to the MH available place
Add an arc from the Activity complete transition to the ending tlocation hasT place

Transfer
Add a transition (type = 1)
Setup pre-conditions
Add a Decision input place (type = 4)
Add an arc from the decision input place to the transition
Add an event transition, message = XFER,starting fpl, ending fpl
Add an arc from the event transition to the decision input place
Add an arc from the starting fpl has part place to the transition
Add an arc from the ending fpl available place to the transition
Add an arc from the MH available place to the transition
Add Activity (fpl_to_WS, XFER, starting fpl, ending fpl, Type= %i,Node= %i)
Setup post conditions
Add an arc from the Activity complete transition to the starting fpl available place
Add an arc from the Activity complete transition to the MH available place
Add an arc from the Activity complete transition to the ending fpl has part place

7. Process storage workstation movement graph arcs
Load
Add a transition (type = 1)
Setup pre-conditions
Add a Decision input place (type = 4)

Add an arc from the decision input place to the transition
Add an event transition, message = LOAD,starting tlocation, ending fpl
Add an arc from the event transition to the decision input place
Add an arc from the ending fpl available place to the transition
Add an arc from the starting tlocation hasT place to the transition
Add an arc from the MH available place to the transition
Add Activity (fpl_to_WS, LOAD,tloc,%I, fpl, Type= %i,Node= %i)
Setup post conditions
Add an arc from the Activity complete transition to the fpl has part place
Add an arc from the Activity complete transition to the MH available place
Add an arc from the Activity complete transition to the starting tlocation hasT place

Unload
Add a transition (type = 1)
Setup pre-conditions
Add a Decision input place (type = 4)
Add an arc from the decision input place to the transition
Add an event transition, message = UNLOAD,starting fpl, ending tlocation
Add an arc from the event transition to the decision input place
Add an arc from the starting fpl has part place to the transition
Add an arc from the ending tlocation hasT place to the transition
Add an arc from the MH available place to the transition
Add Activity (fpl_to_WS, UNLOAD, fpl, tloc,%I, Type= %i,Node= %i)
Setup post conditions
Add an arc from the Activity complete transition to the fpl available place
Add an arc from the Activity complete transition to the MH available place
Add an arc from the Activity complete transition to the ending tlocation hasT place

8. Add Tokens for transporters

9. Add Tokens for equipment availability

10. Add Tokens for Parts and Part carriers

Process Plan Conversion Algorithm

All arcs in this section are type 0. All tokens are type 0 (information). No tokens are entered during the Petri net creation process. The tokens must be taken from the users parts inventory.

1. For each node in the process plan add a standard place to the Petri net. Add an entry to the PartIndex table.

2. If a node is **NOT** a default start node for a process plan then:
A. If a process node has only one arc leaving it then
Add a transition
Add an arc from the process node to the transition
Add an input place with its associated event triggered transition. The event is a "PROCESS COMPLETED" message from the appropriate workstation controller.
Add an arc from the input place to the transition.
Add an activity. Message is "TRANSFORM"

Add an arc from the output transition to the process node at the head of the process plan arc

B. If a process node has more than one arc leaving it then
Add a transition
Add an arc from the process node to the transition
Add an input place with its associated event triggered transition. The event is a "PROCESS COMPLETED" message from the appropriate workstation controller.
Add an arc from the input place to the transition.
Add a standard place that represents "waiting for a decision"
Add an arc from the transition to the "waiting for a decision" place
For each arc leaving the process node
Add a transition
Add an arc from the "waiting for a decision" node to the transition
Add a decision input place with its associated event triggered transition. The event is a "TRANSFORM" message from the neural net.
Add an arc from the input place to the transition.
Add an activity. Message is "TRANSFORM"
Add an arc from the output transition to the process node at the head of the process plan arc

3. If a node is a default start node for a process plan then:
For each arc leaving the process node:
For each storage location in the model:
Add a transition
Add an arc from the process node to the transition
Add a decision input place with its associated event triggered transition. The event is a "START" message from the neural net.
Add an arc from the input place to the transition.
Add an activity. Message is "TRANSFORM"
Add an arc from the output transition to the process node at the head of the process plan arc

APPENDIX I

PETRI NET MARKING ALGORITHM FOR STATUS MATRIX CONSTRUCTION

The status matrix row information table is created by adding an entry for each entry in the Petri net node table that is marked as a status matrix row.

The status matrix column information table is created by adding one entry for each transporter type, adding an entry for each part carrier type, and adding one column for each entry in the Petri net node table that is marked as a status matrix column.

Marking the entries in the Petri net node table

1. For every TLocation, there is a node representing a transporter occupying the TLocation, mark this node as an SMRow that does not get checked for deadlock.
2. For every storage workstation there is a node that represents the parts in the location, mark this node as an SMRow that does not get checked for deadlock.
3. For every fixed part location that is not part of a storage work station there is a node that represents the location is occupied by a part, mark this node as an SMRow that does not get checked for deadlock.
4. For every fixed part location that is associated with a material processor, there is a node that represents the material processor performing an operation on a part, mark this node as an SMRow that does get checked for deadlock.
5. For each transporter graph movement arc there is a node that represents the transporter moving across the arc, mark this node as an SMRow that does get checked for deadlock.
6. Process workstation arcs are combined such that mobile part locations are not kept distinct at the cell controller level. For each load point associated with a processing workstation there exists a set of combined arcs representing movement from the load point to the various fixed part locations in the processing workstation. For each combined arc, there is a node representing the fact the transporter is involved in a load operation, mark this node as an SMRow that gets checked for deadlock. Further, for each combined arc there is also a node representing that a part is involved in a load operation, mark this node as an SMRow that gets checked for deadlock.
7. For each unload point associated with a processing workstation there exists a set of combined arcs representing movement from the various fixed part locations in the processing workstation to the load point. For each combined arc, there is a node representing the fact the transporter is involved in an unload operation, mark this node as an SMRow that gets checked for deadlock. Further, for each combined arc there is also a node representing that a part is involved in an unload operation, mark this node as an SMRow that gets checked for deadlock.
8. For each possible transfer within a workstation, there will be a node representing that a part is transferring, mark this node as an SMRow that gets checked for deadlock.
9. Storage workstation arcs are also combined at the cell level and transfers within the storage workstation are not considered. . For each load point associated with a storage workstation there exists a set of combined arcs representing movement from the load point to the fixed part location representing the storage workstation. For each combined arc, there is a node representing the fact the transporter is involved in a load operation, mark this node as an SMRow that gets checked for deadlock. Further, for each combined arc there is also a node representing that a part is involved in a load operation, mark this node as an SMRow that gets checked for deadlock.
10. For each unload point associated with a storage workstation there exists a set of combined arcs representing movement from the fixed part location representing the storage workstation to the unload point. For each combined arc, there is a node representing the fact the transporter is

involved in an unload operation, mark this node as an SMRow that gets checked for deadlock. Further, for each combined arc there is also a node representing that a part is involved in an unload operation, mark this node as an SMRow that gets checked for deadlock.

11. For each process plan node there is a corresponding Petri net node, mark this node as an SMColumn.
12. For each process plan node that has multiple arcs leaving the node there is a Petri net node that represents the process being complete and a decision regarding which arc in the process plan to take, mark this node as an SMColumn.

APPENDIX J

DEADLOCK AND STALL RECOVERY

Deadlocks and stalls were divided into four major categories: 1) a processing workstation circular wait, 2) a part blocked from exiting a processing workstation, 3) a part blocked from exiting a storage workstation, and 4) a part in the transportation system. These categories were then subdivided giving the eighteen categories listed in Table 136. The "ID No." is the value returned from the deadlock classification function.

Table 136 Deadlock and Stall Categories

Id No.	Category	Description
1	1A	Processing WS circular wait, no buffers in the WS
2	1B	Processing WS circular wait, an empty buffer in the WS
3	1C1	Processing WS circular wait, all buffers full at least one part in the WS wants to exit the WS
4	1C2	Processing WS circular wait, all buffers full all parts want to remain in the WS
5	2A	Processing WS, no transporters at the WS unload points
6	2B	Processing WS, no transporter capacity at primary unload point, available capacity at a secondary unload point
7	2C	Processing WS, type 4 unload arc, type 3 space available no type 4 space available
8	2D	Processing WS, type 4 unload arc, no type 3 space and no type 4 space available
9	2E	Processing WS, type 3 unload arc, no type 3 space available
10	3	Storage WS, no unload logic implemented for a partially processed part
11	4	Part located on a transporter that is not located at the proper WS load point
12	3A	Storage WS, no transporters at the WS unload points
13	3B	Storage WS, no transporter capacity at primary unload point, available capacity at a secondary unload point
14	3C	Storage WS, type 4 unload arc, type 3 space available no type 4 space available
15	3D	Storage WS, type 4 unload arc, no type 3 space and no type 4 space available
16	3E	Storage WS, type 3 unload arc, no type 3 space available
17	2F	Processing WS, no transporter at primary unload point, transporter without available capacity at secondary unload point
18	3F	Storage WS, no transporter at primary unload point, transporter without available capacity at secondary unload point

Recovery Procedures

The initial exemplar creation process generated a set of alternative movement paths for transfers between machines. These paths involved a direct transfer between machines, if possible, transfer from the machine to a buffer and from the buffer to the second machine, if the workstation contains buffers, and an unload operation followed by movement from the unload point to a load point and then a load operation to the second machine. These paths were then prioritized and the threshold levels set on the layer 2 neural net node associated with the path such that only the highest priority path could be activated.

A large number of the recovery actions will require unloading a part from a workstation. Transporter capacity must be available at a workstation unload point to allow a part to be unloaded. In fact, the lack of available transport capacity is the cause of all type 2 and most type 3 stalls. Most recovery actions will consist of finding a transporter with available capacity and moving it to the workstation unload point. This may require moving other transporters to clear a path for the transporter with capacity to reach the unload point. If there are no transporters with available capacity, it will be necessary to move a transporter with unfinished parts to a storage workstation load point and place a part into storage to create available transport capacity that can then be moved to the workstation that must be unloaded.

In recovering from deadlocks and stalls, processing workstations were assigned the highest priority, parts in the transportation system the second highest priority and parts in storage workstations the lowest priority.

1A

To recover from a circular wait in a workstation with no buffers, a part must be unloaded to make space to move other parts that are in the workstation. The first step is to check for the availability of transporter capacity located at an unload point for the workstation. If transporter capacity is present then the neural net logic must be modified to cause the appropriate unload command to be activated. If

transporter capacity is not present then a transporter with available capacity must be found and moved to one of the workstations unload points.

1B

To recover from a circular wait in a workstation with an empty buffer, a part is moved from one of the machines involved in the circular wait to the buffer allowing the other parts to move as they desire and then the part in the buffer will move to the machine it desires from the buffer when it becomes available.

1C1

In this case, the circular wait is secondary to the problem of a part wanting to exit the workstation. The failure of the part to leave the workstation makes this problem equivalent to a type 2 stall. Removing the part may transform the situation into a type 1B deadlock where a buffer in the workstation may be used to solve the circular wait. If the part that wants to exit the workstation is not located in a buffer, then after the part is removed the workstation the situation may remain a 1C1, if there were multiple parts that wanted to leave the workstation or it may be transformed into a type 1C2. It is also possible that removal of a part from a machine will allow a part in a buffer to be transferred to the machine resulting in a indirect conversion to a type 1B stall. The stall will not be corrected until the parts that have begun moving or processing have completed all available processing.

1C2

When all of the parts in the workstation wish to remain in the workstation, one of the parts involved in the circular wait must be removed to allow the other parts to progress. This part will be a part located on one of the machines. It is important to note that parts in a buffer can not be involved in a circular wait because none of the parts on the machines desire a buffer as their next destination. They either wish to move to another machine in the workstation or to leave the workstation. Parts wanting to leave the workstation wish to move directly to an unload point not to a buffer. When resolving a circular wait, it is preferable to give parts on other machines priority over parts in buffers after removing a part

from a machine. This will allow the maximum number of parts to progress forward. A part will not necessarily move to a buffer from a machine to allow other parts requiring the machine to progress.

2A

Find a transporter with available capacity and move it to a workstation unload point.

2B

Add neural net logic to select an unload for the secondary unload point that has the available transport capacity.

2C

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away. In this case the transporter currently at the unload point has the wrong type of capacity. This will happen when the workstation being unloaded requires that a part carrier be on the transporter and no part carrier is there.

2D

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away.

2E

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away. This will occur either if the transporter is filled to capacity or it has a part carrier that is able to accept a part, but the workstation that needs to be unloaded unloads a part carrier with the part.

2F

Find a transporter with available capacity and move it to a workstation unload point. The transporter currently at the unload point may or may not need to be moved depending on the configuration of the transportation system.

3

Add neural net logic to allow the unload command to be activated.

3A

Find a transporter with available capacity and move it to a workstation primary unload point for the part type and node combination to be unloaded. There may not be any unload logic present to unload the part to a non-primary unload point.

3B

Add neural net logic to select an unload for the secondary unload point that has the available transport capacity.

3C

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away. In this case the transporter currently at the unload point has the wrong type of capacity. This will happen when the workstation being unloaded requires that a part carrier be on the transporter and no part carrier is there.

3D

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away.

3E

Find a transporter with available capacity and move it to a workstation unload point. Move the transporter currently at the unload point away. This will occur either if the transporter is filled to capacity or it has a part carrier that is able to accept a part, but the workstation that needs to be unloaded unloads a part carrier with the part.

3F

Find a transporter with available capacity and move it to a workstation unload point. The transporter currently at the unload point may or may not need to be moved depending on the configuration of the transportation system.

4

Add neural net logic to move the transporter toward the required workstation. It may be necessary to add neural net logic to move transporters that are blocking the transporter with the part.

APPENDIX K

ZIP FILE CONTENTS

Two zip files are included with this dissertation: releasecandidate9.zip and finaltestcasemodels.zip. The releasecandidate9.zip file contains Visual C++ source code for the cell controller organized into three subdirectories: 1) timedCellController, 2) fixedrules, and 3) exemplars. The timedCellController directory contains the code for the actual controller. The fixedrules directory contains the code to create the initial input for timedCellController from the user model. The exemplars directory contains the code used to generate the control logic. It modifies the files created by the code in the fixed rules directory and creates additional files used by timedCellController.

The finaltestcasemodels.zip file contains the user input models for the test cases used in this dissertation.

VITA

Wesley Dane Scott

Permanent Address:

1051 N Dean St., Coquille, OR 97423

Educational Background:

B.S., Chemical Engineering, Oregon State University, 1985

M.S.E., Mechanical Engineering, Purdue University, 1992